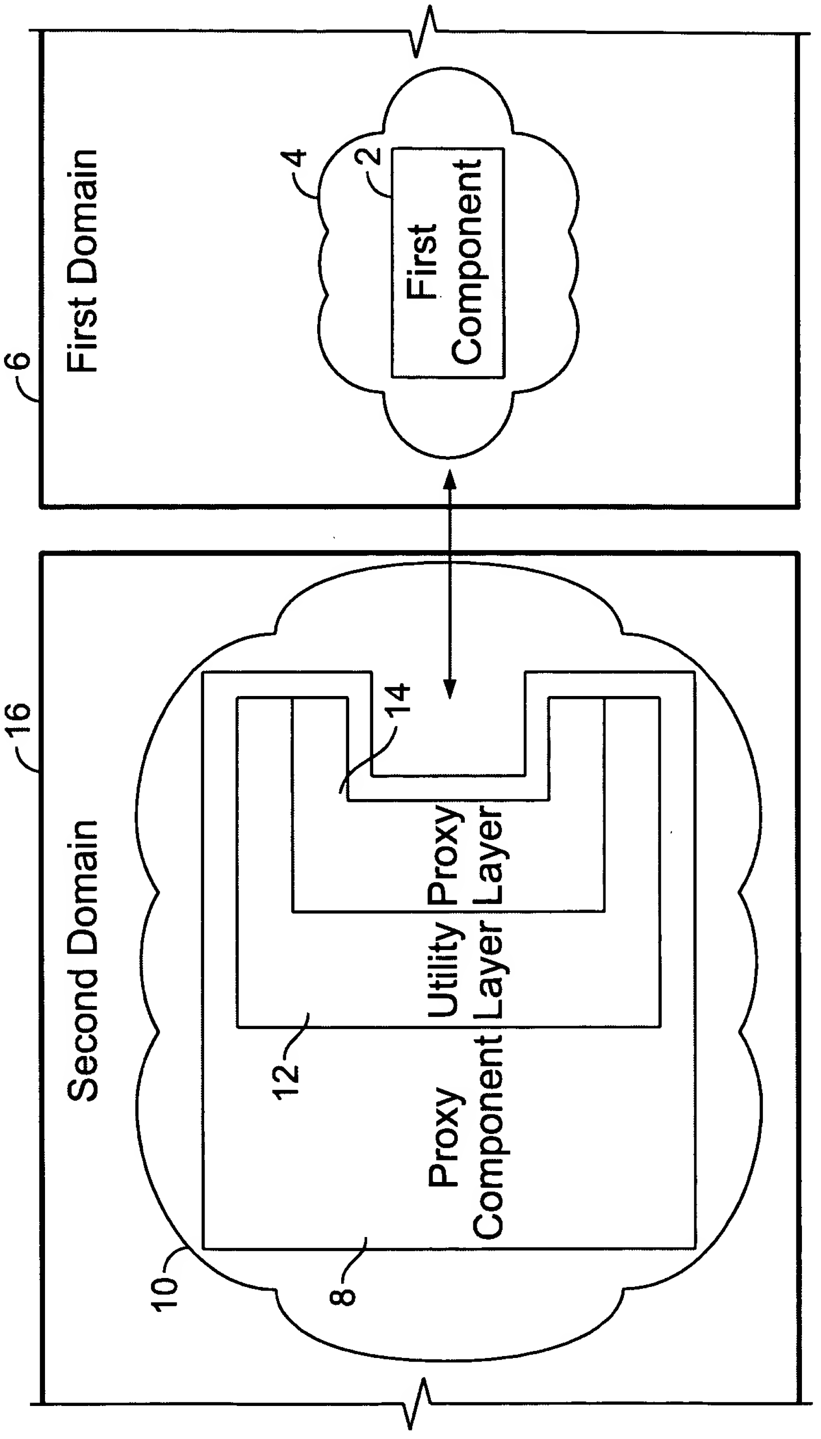

First Domain
6
4
2
First Component
Second Domain
16
14
Proxy Layer
Utility Layer
12
Proxy Component
10
8

FIG. 1

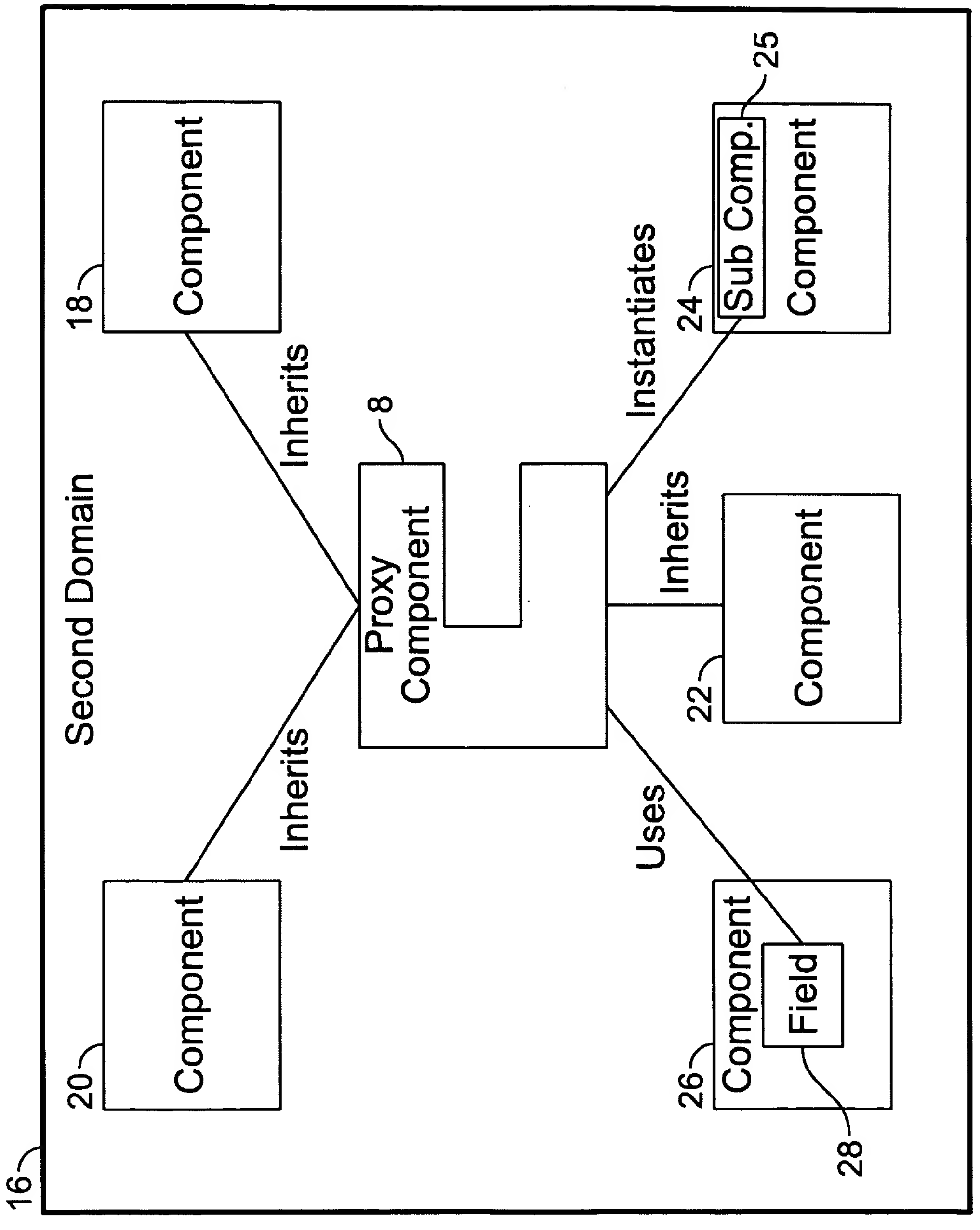
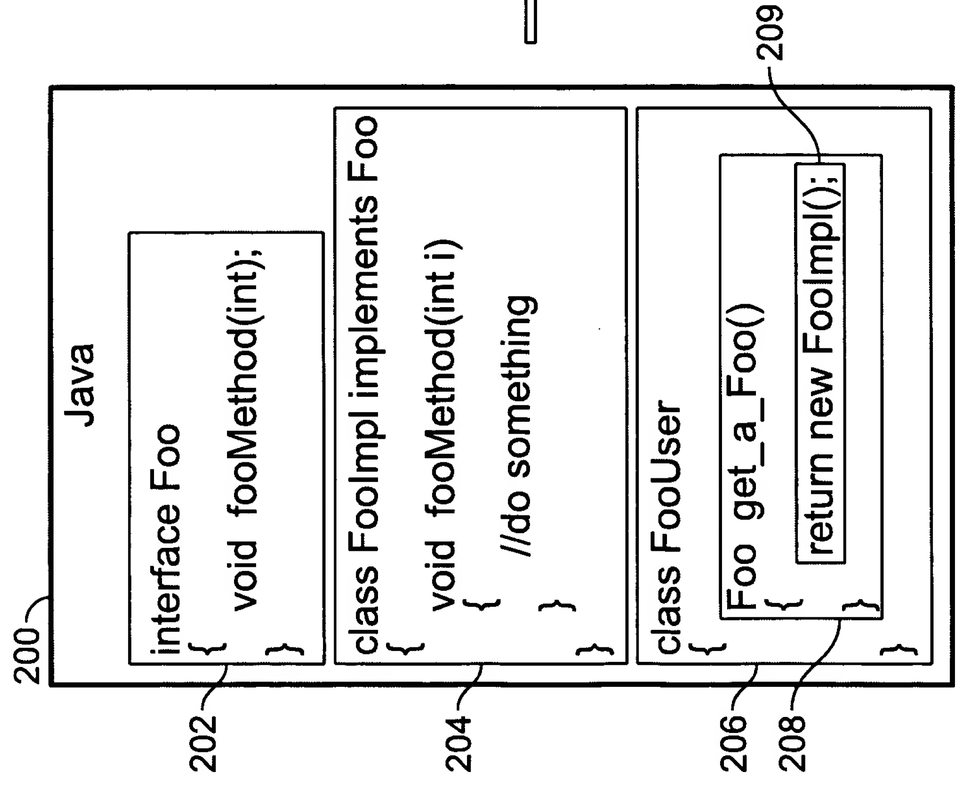
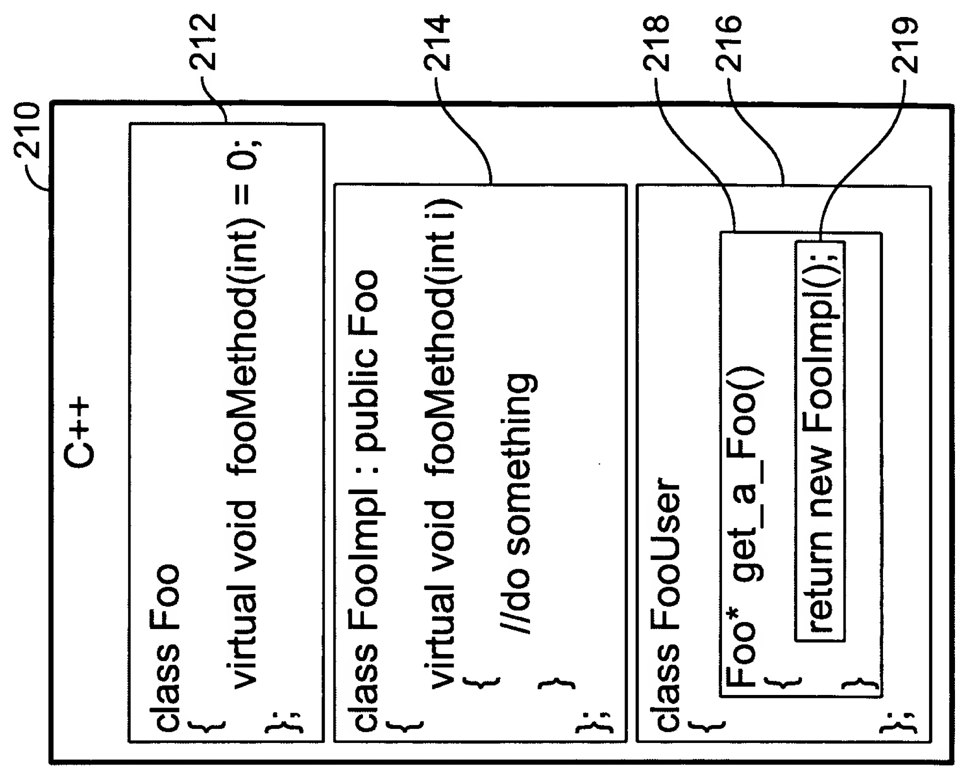

Second Domain
16
Component
18
Component
20
Inherits
Inherits
Proxy Component
8
Instantiates
Sub Comp.
24
25
Component
Inherits
Component
22
Uses
Component
26
Field
28

FIG. 2



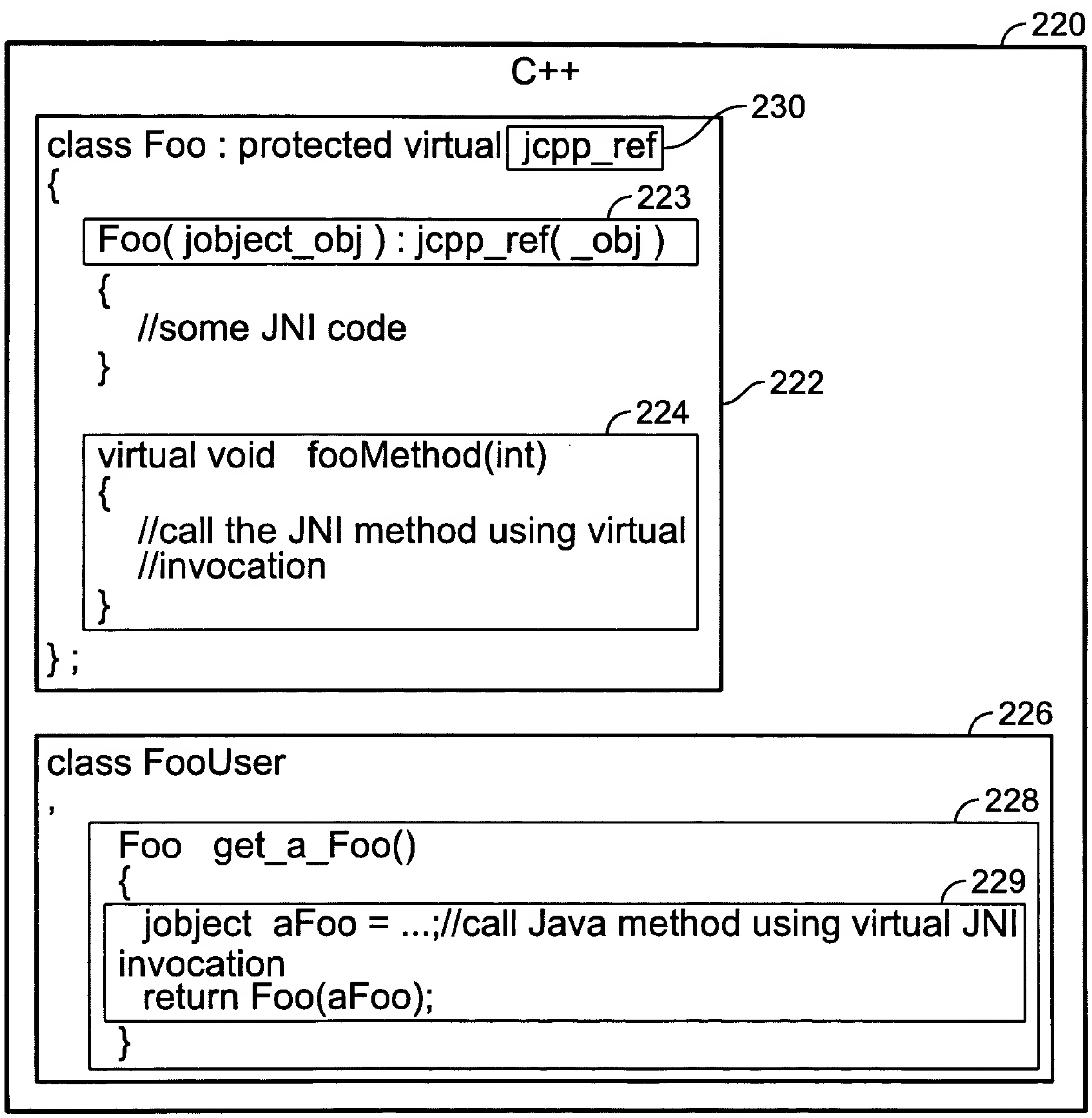
```
interface Foo
{
    void  fooMethod(int);
}
class FooImpl implements Foo
{
    void  fooMethod(int i)
    {
        //do something
    }
}
class FooUser
{
    Foo  get_a_Foo()
    {
        return new FooImpl();
    }
}
```

FIG. 3



```
class Foo
{
    virtual void  fooMethod(int) = 0;
};
class FooImpl : public Foo
{
    virtual void  fooMethod(int i)
    {
        //do something
    }
};
class FooUser
{
    Foo*  get_a_Foo()
    {
        return new FooImpl();
    }
};
```

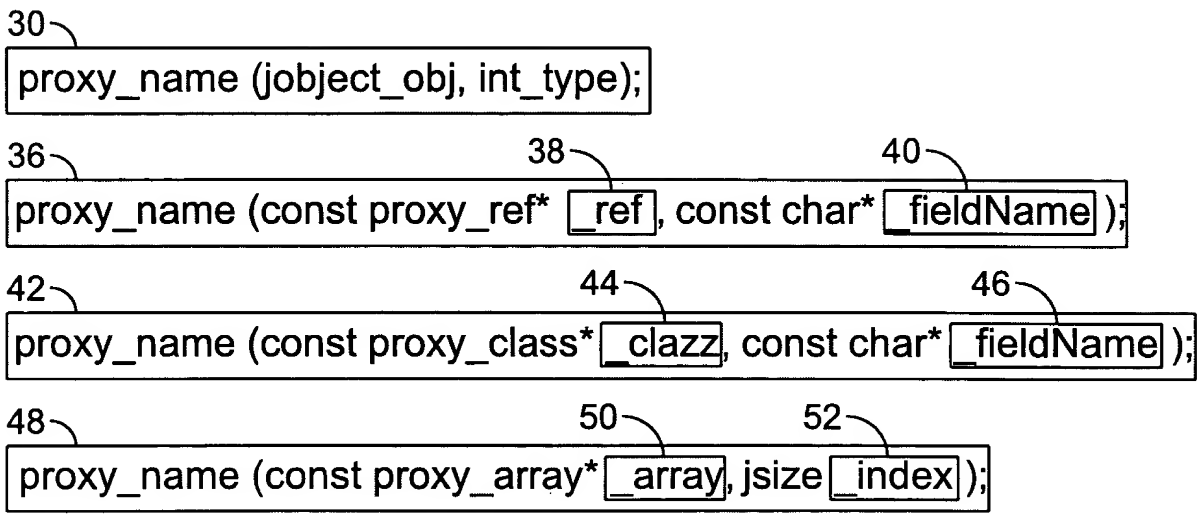
FIG. 4



```
class Foo : protected virtual jcpp_ref
{
    Foo( jobject_obj ) : jcpp_ref( _obj )
    {
        //some JNI code
    }

    virtual void   fooMethod(int)
    {
        //call the JNI method using virtual
        //invocation
    }
};
```

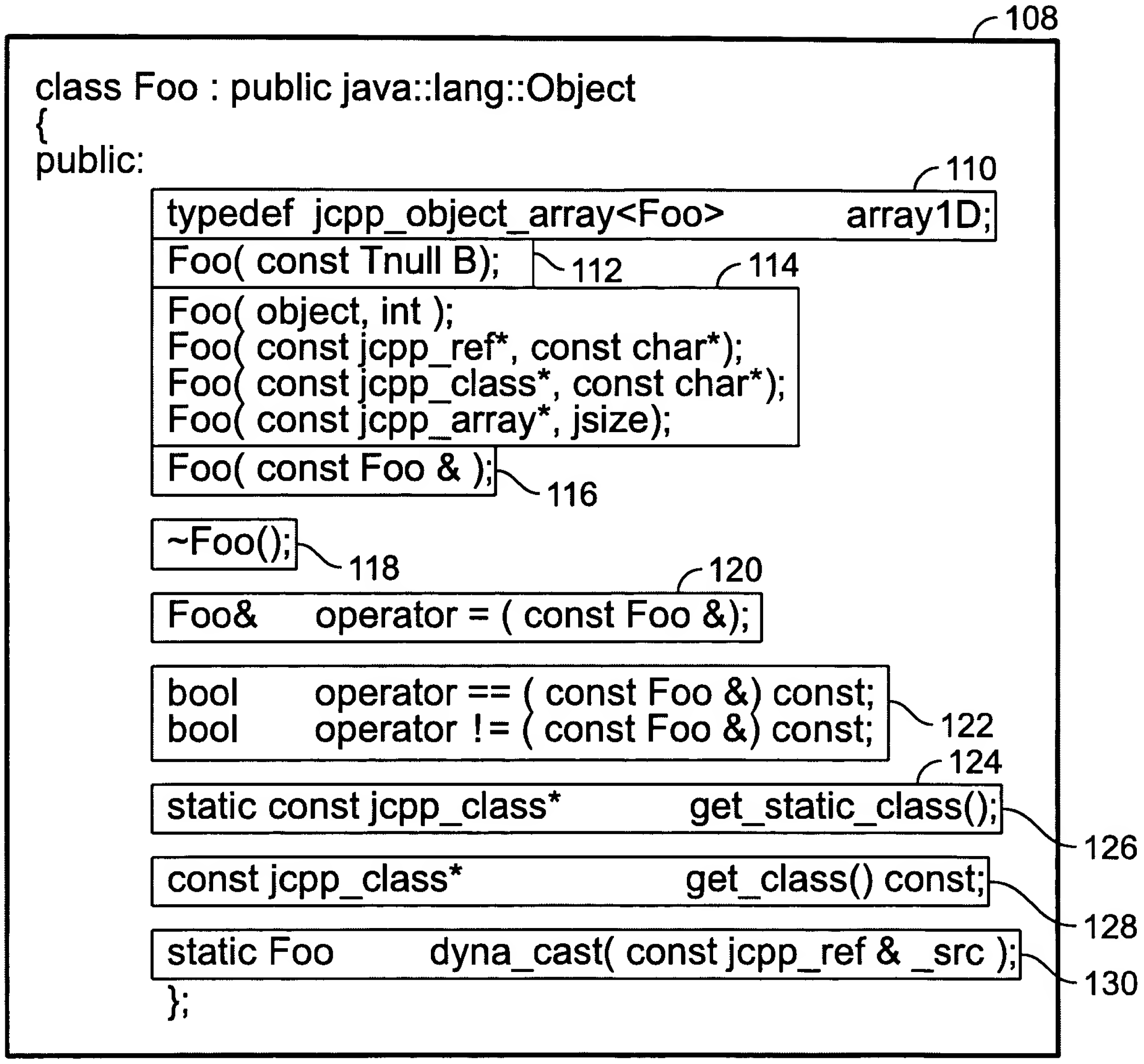
```
class FooUser
,
    Foo   get_a_Foo()
    {
      jobject  aFoo = ...;//call Java method using virtual JNI
    invocation
      return Foo(aFoo);
    }
```

FIG. 5



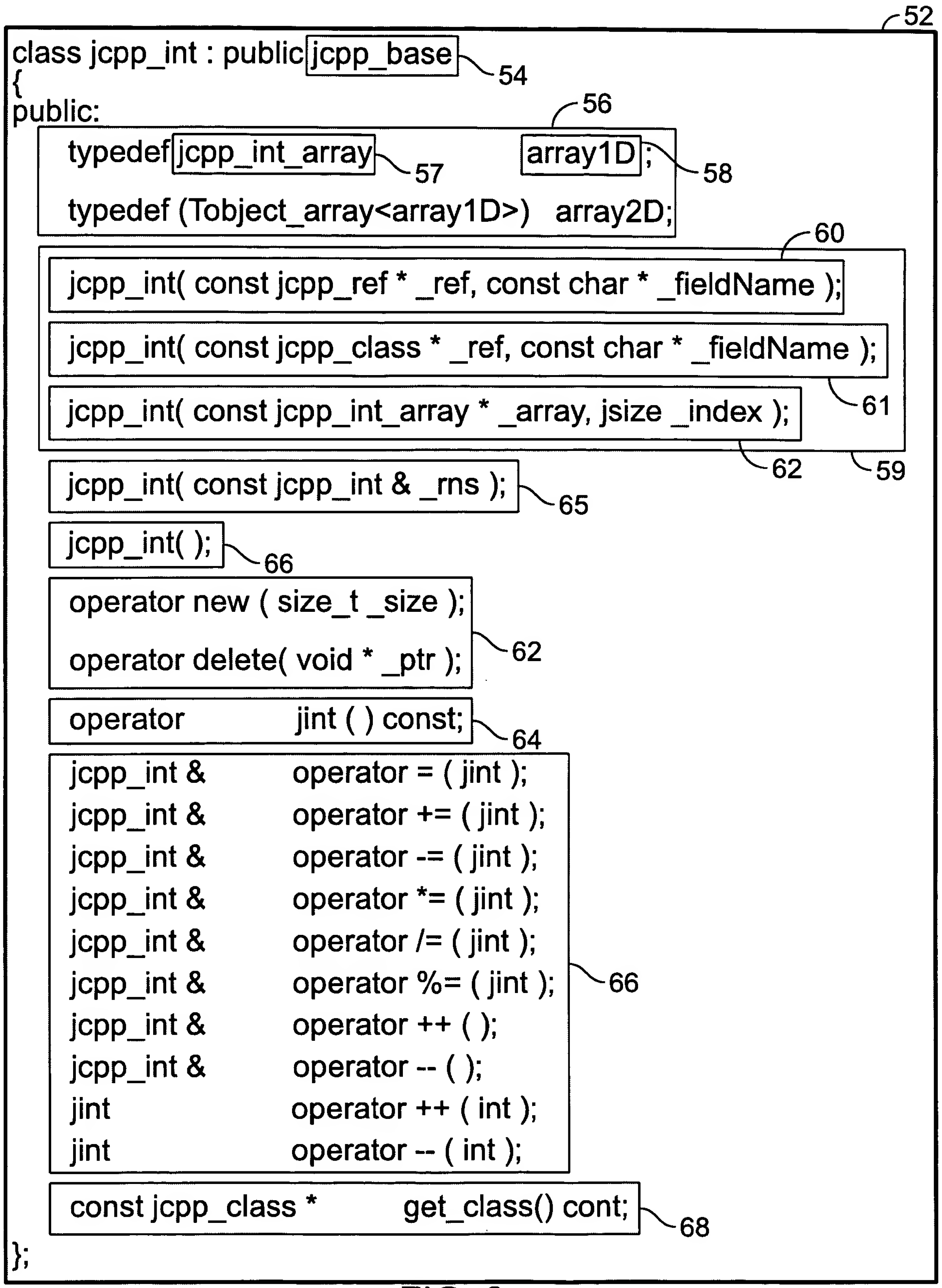
```
proxy_name (jobject_obj, int_type);
proxy_name (const proxy_ref* _ref, const char* _fieldName );
proxy_name (const proxy_class* _clazz, const char* _fieldName );
proxy_name (const proxy_array* _array, jsize _index );
```

FIG. 6



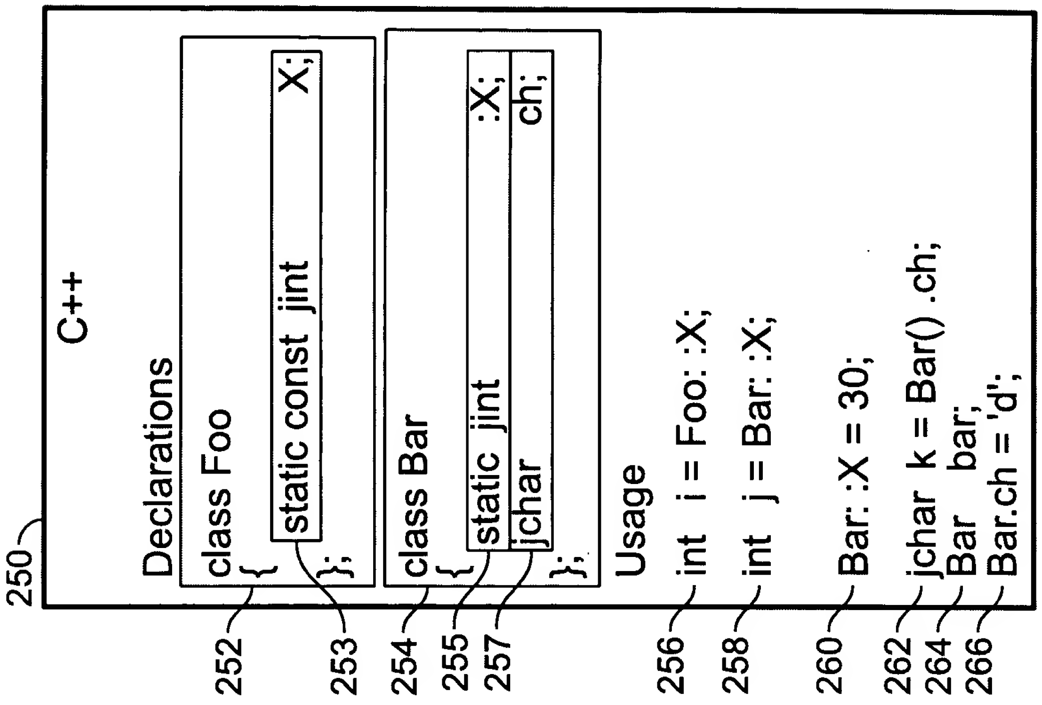
```
class Foo : public java::lang::Object
{
public:
        typedef  jcpp_object_array<Foo>          array1D;
        Foo( const Tnull B);
        Foo( object, int );
        Foo( const jcpp_ref*, const char*);
        Foo( const jcpp_class*, const char*);
        Foo( const jcpp_array*, jsize);
        Foo( const Foo & );

        ~Foo();

        Foo&     operator = ( const Foo &);

        bool       operator == ( const Foo &) const;
        bool       operator != ( const Foo &) const;

        static const jcpp_class*           get_static_class();

        const jcpp_class*                  get_class() const;

        static Foo        dyna_cast( const jcpp_ref & _src );
        };
```

FIG. 7



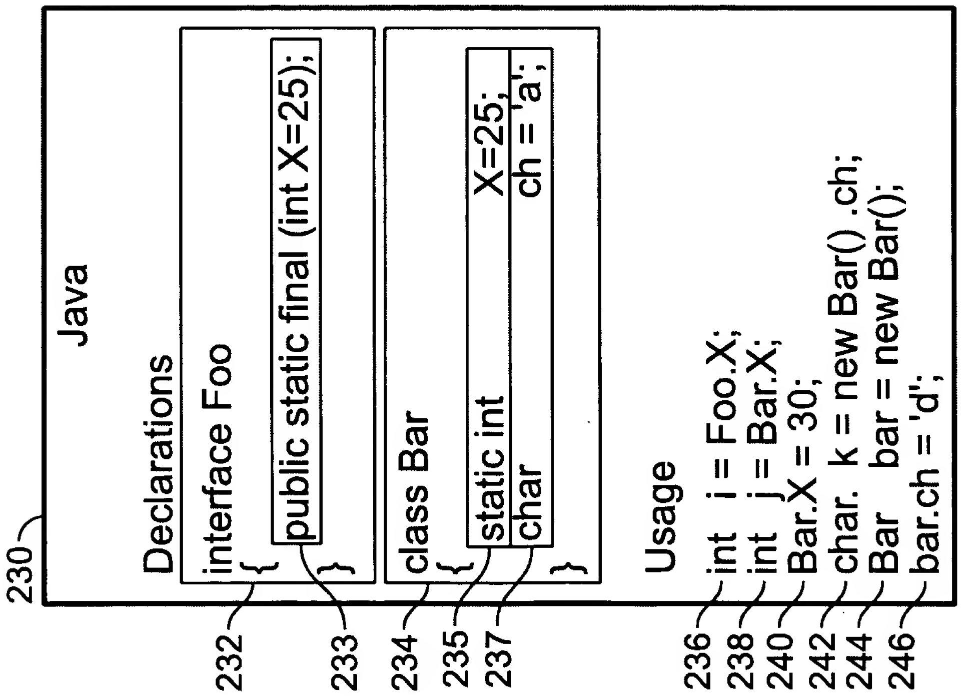
```
class jcpp_int : public jcpp_base
{
public:
    typedef jcpp_int_array          array1D;
    typedef (Tobject_array<array1D>)   array2D;

    jcpp_int( const jcpp_ref * _ref, const char * _fieldName );
    jcpp_int( const jcpp_class * _ref, const char * _fieldName );
    jcpp_int( const jcpp_int_array * _array, jsize _index );

    jcpp_int( const jcpp_int & _rns );

    jcpp_int( );

    operator new ( size_t _size );
    operator delete( void * _ptr );

    operator          jint ( ) const;

    jcpp_int &        operator = ( jint );
    jcpp_int &        operator += ( jint );
    jcpp_int &        operator -= ( jint );
    jcpp_int &        operator *= ( jint );
    jcpp_int &        operator /= ( jint );
    jcpp_int &        operator %= ( jint );
    jcpp_int &        operator ++ ( );
    jcpp_int &        operator -- ( );
    jint              operator ++ ( int );
    jint              operator -- ( int );

    const jcpp_class *       get_class() cont;
};
```

FIG. 8



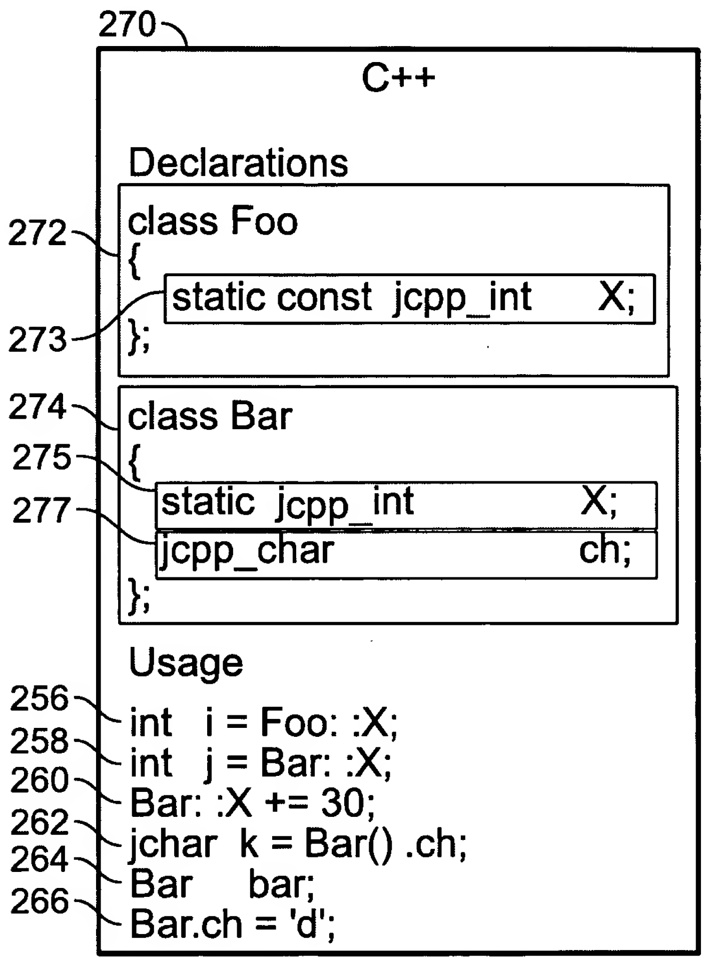
```
C++
Declarations
class Foo
{
    static const jint        X;
};
class Bar
{
    static  jint            :X;
    jchar                    ch;
};
Usage
int  i = Foo: :X;
int  j = Bar: :X;

Bar: :X = 30;
jchar  k = Bar() .ch;
Bar    bar;
Bar.ch = 'd';
```

FIG. 10



```
Java
Declarations
interface Foo
{
    public static final (int X=25);
}
class Bar
{
    static int          X=25;
    char                ch = 'a';
}
Usage
int  i = Foo.X;
int  j = Bar.X;
Bar.X = 30;
char.  k = new Bar() .ch;
Bar    bar = new Bar();
bar.ch = 'd';
```

FIG. 9



```
class Foo
{
    static const  jcpp_int      X;
};

class Bar
{
    static  jcpp_int              X;
    jcpp_char                     ch;
};
```

```
int   i = Foo: :X;
int   j = Bar: :X;
Bar: :X += 30;
jchar  k = Bar() .ch;
Bar     bar;
Bar.ch = 'd';
```

FIG. 11

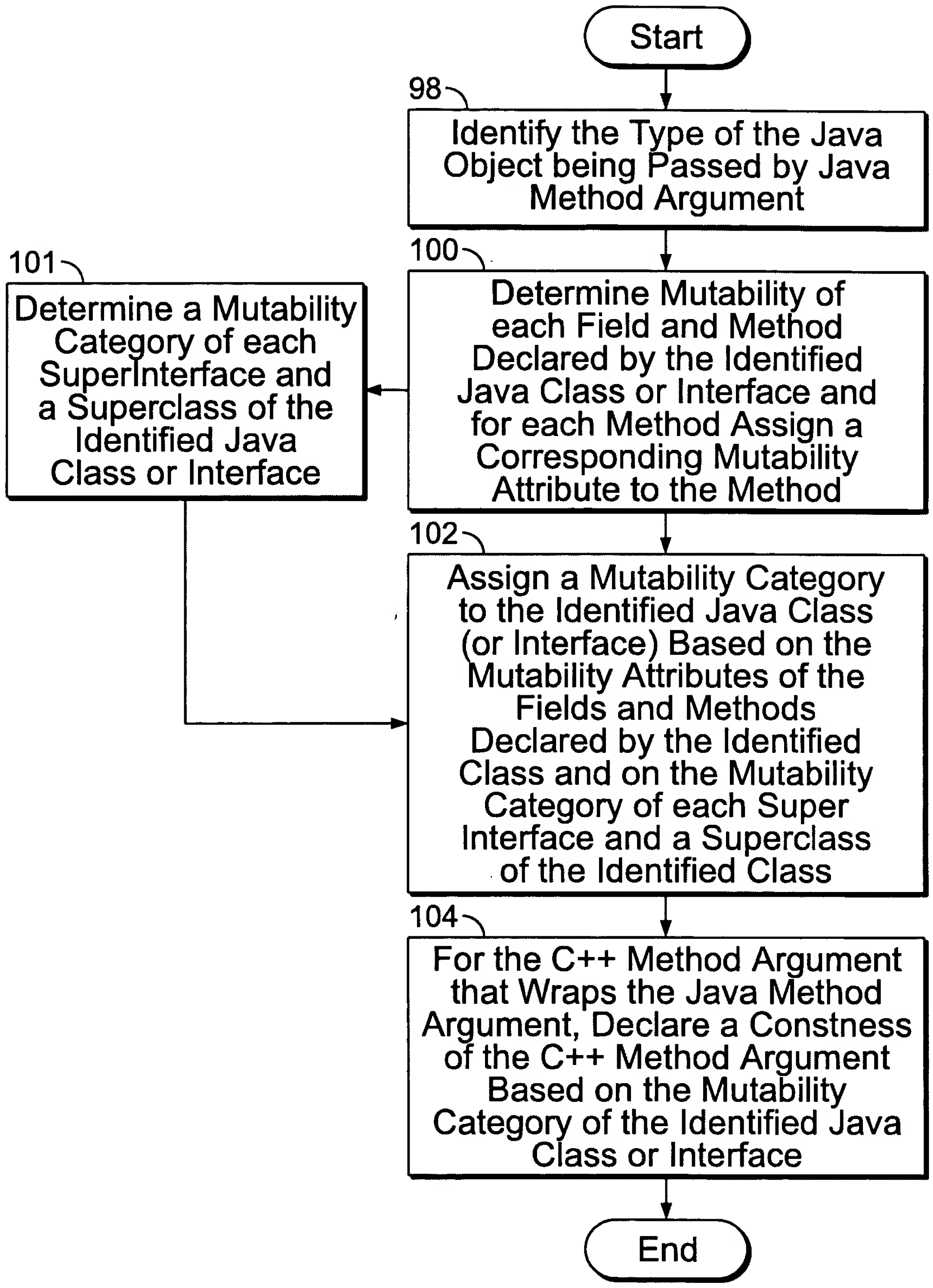

Start
98
Identify the Type of the Java Object being Passed by Java Method Argument
100
Determine Mutability of each Field and Method Declared by the Identified Java Class or Interface and for each Method Assign a Corresponding Mutability Attribute to the Method
101
Determine a Mutability Category of each SuperInterface and a Superclass of the Identified Java Class or Interface
102
Assign a Mutability Category to the Identified Java Class (or Interface) Based on the Mutability Attributes of the Fields and Methods Declared by the Identified Class and on the Mutability Category of each Super Interface and a Superclass of the Identified Class
104
For the C++ Method Argument that Wraps the Java Method Argument, Declare a Constness of the C++ Method Argument Based on the Mutability Category of the Identified Java Class or Interface
End

FIG. 12
]

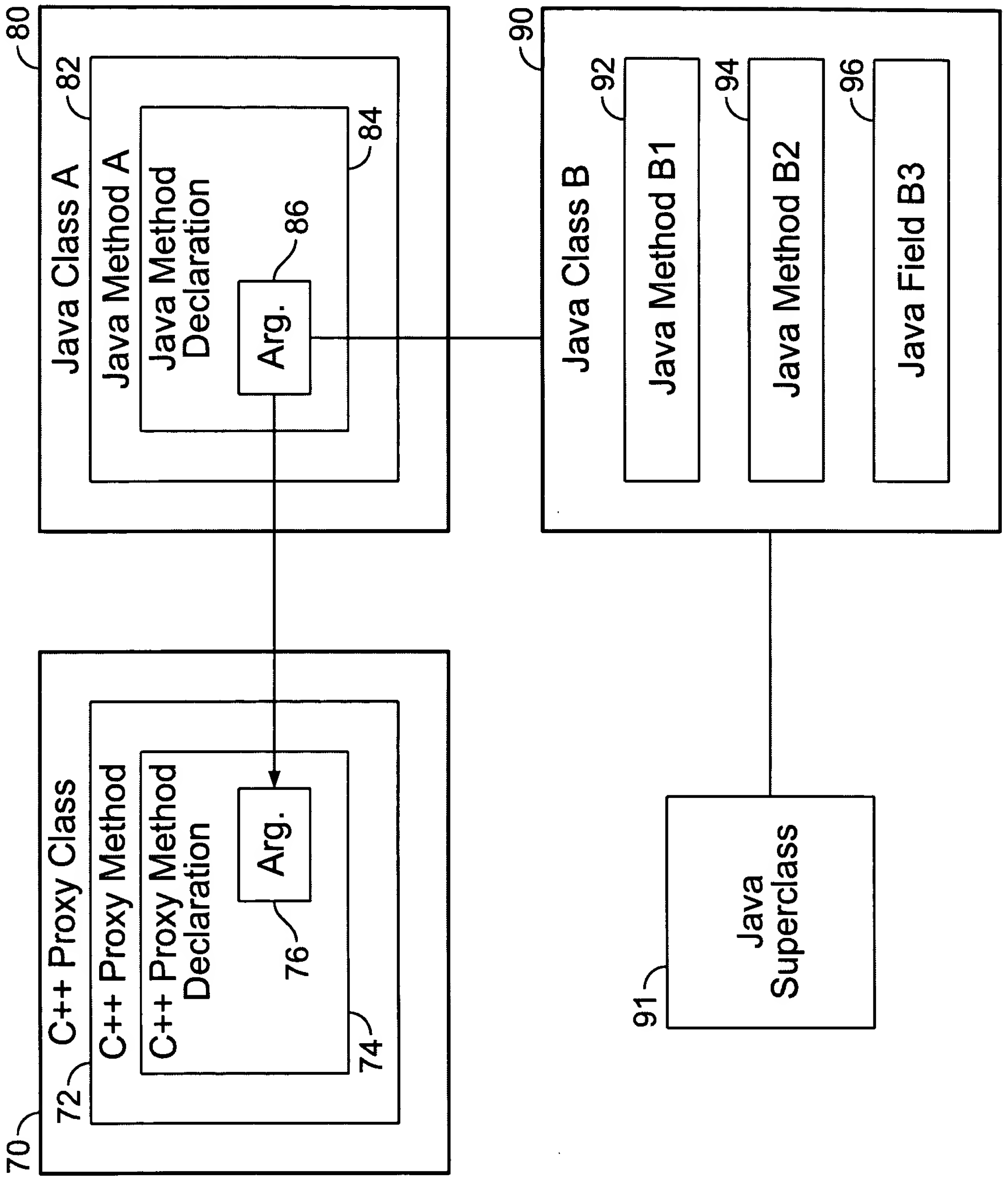

80
Java Class A
82
Java Method A
84
Java Method Declaration
86
Arg.
90
Java Class B
92
Java Method B1
94
Java Method B2
96
Java Field B3
70
C++ Proxy Class
72
C++ Proxy Method
74
C++ Proxy Method Declaration
76
Arg.
91
Java Superclass

FIG. 13

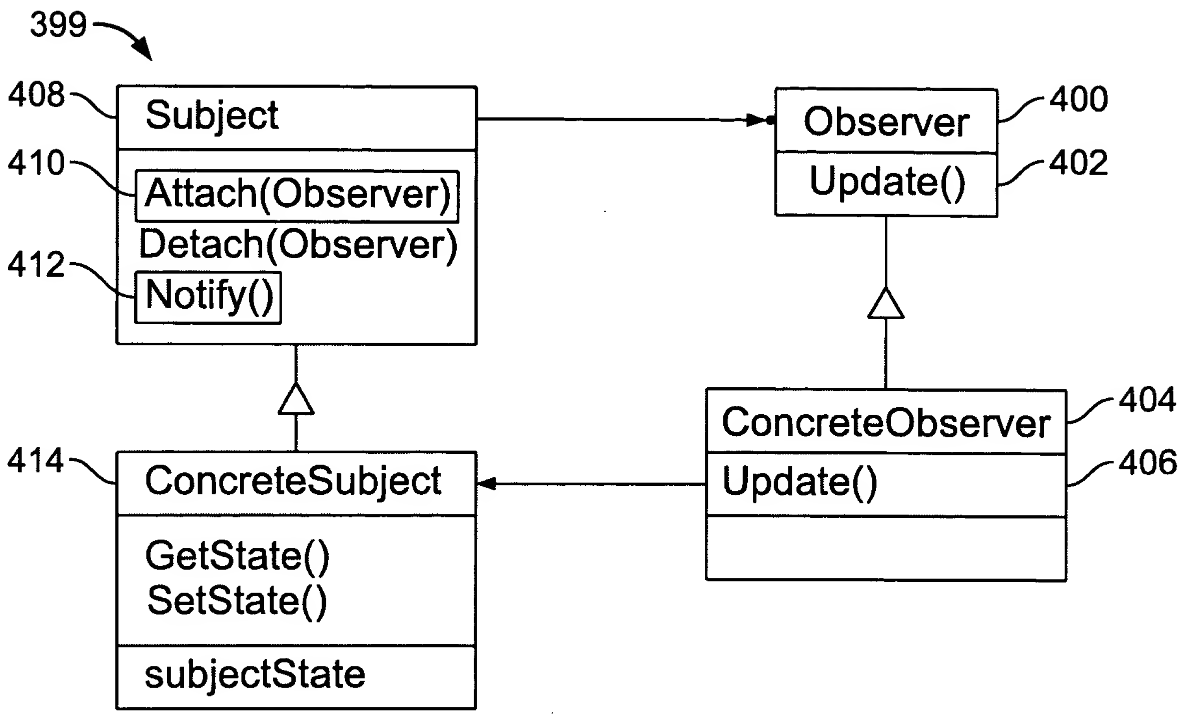

399
408
Subject
410
Attach(Observer)
Detach(Observer)
412
Notify()
Observer
400
Update()
402
ConcreteObserver
404
Update()
406
414
ConcreteSubject
GetState()
SetState()
subjectState

FIG. 14

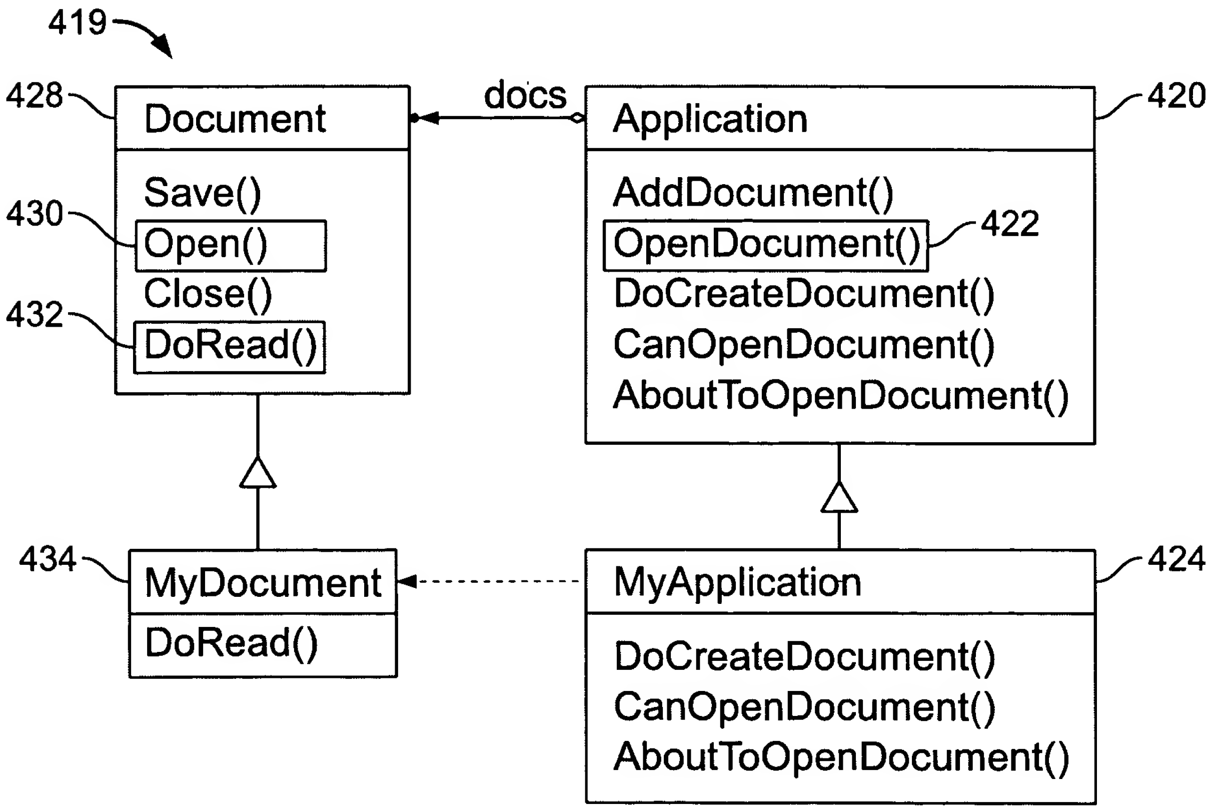

419
428
Document
Save()
430
Open()
Close()
432
DoRead()
docs
Application
420
AddDocument()
OpenDocument()
422
DoCreateDocument()
CanOpenDocument()
AboutToOpenDocument()
434
MyDocument
DoRead()
MyApplication
424
DoCreateDocument()
CanOpenDocument()
AboutToOpenDocument()

FIG. 15

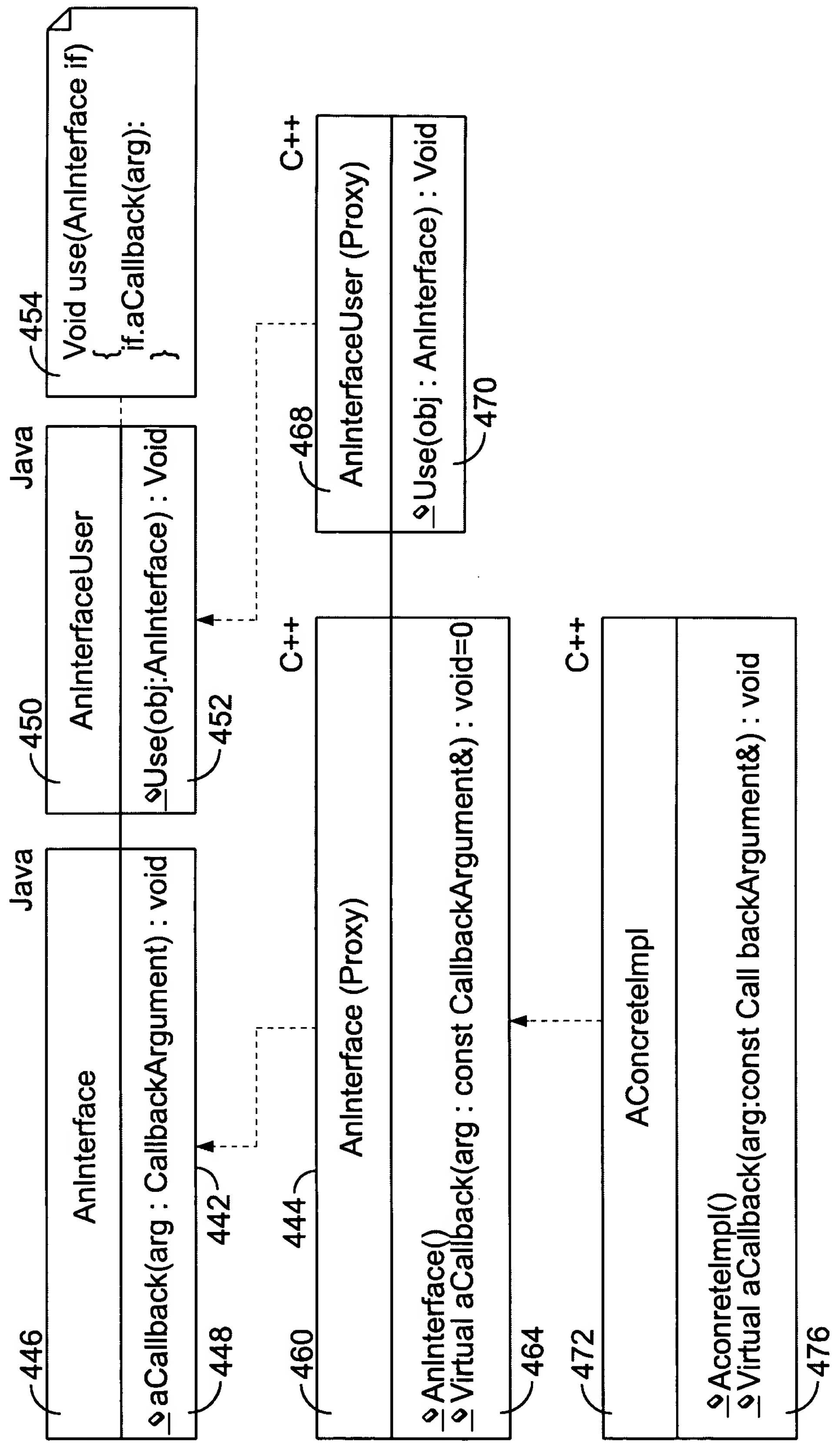

454
Void use(AnInterface if)
{
if.aCallback(arg);
}
Java
450
AnInterfaceUser
Use(obj:AnInterface) : Void
452
Java
446
AnInterface
aCallback(arg : CallbackArgument) : void
448
442
C++
468
AnInterfaceUser (Proxy)
Use(obj : AnInterface) : Void
470
C++
460
AnInterface (Proxy)
AnInterface()
Virtual aCallback(arg : const CallbackArgument&) : void=0
464
444
C++
472
AConcreteImpl
AconreteImpl()
Virtual aCallback(arg:const Call backArgument&) : void
476

FIG. 16

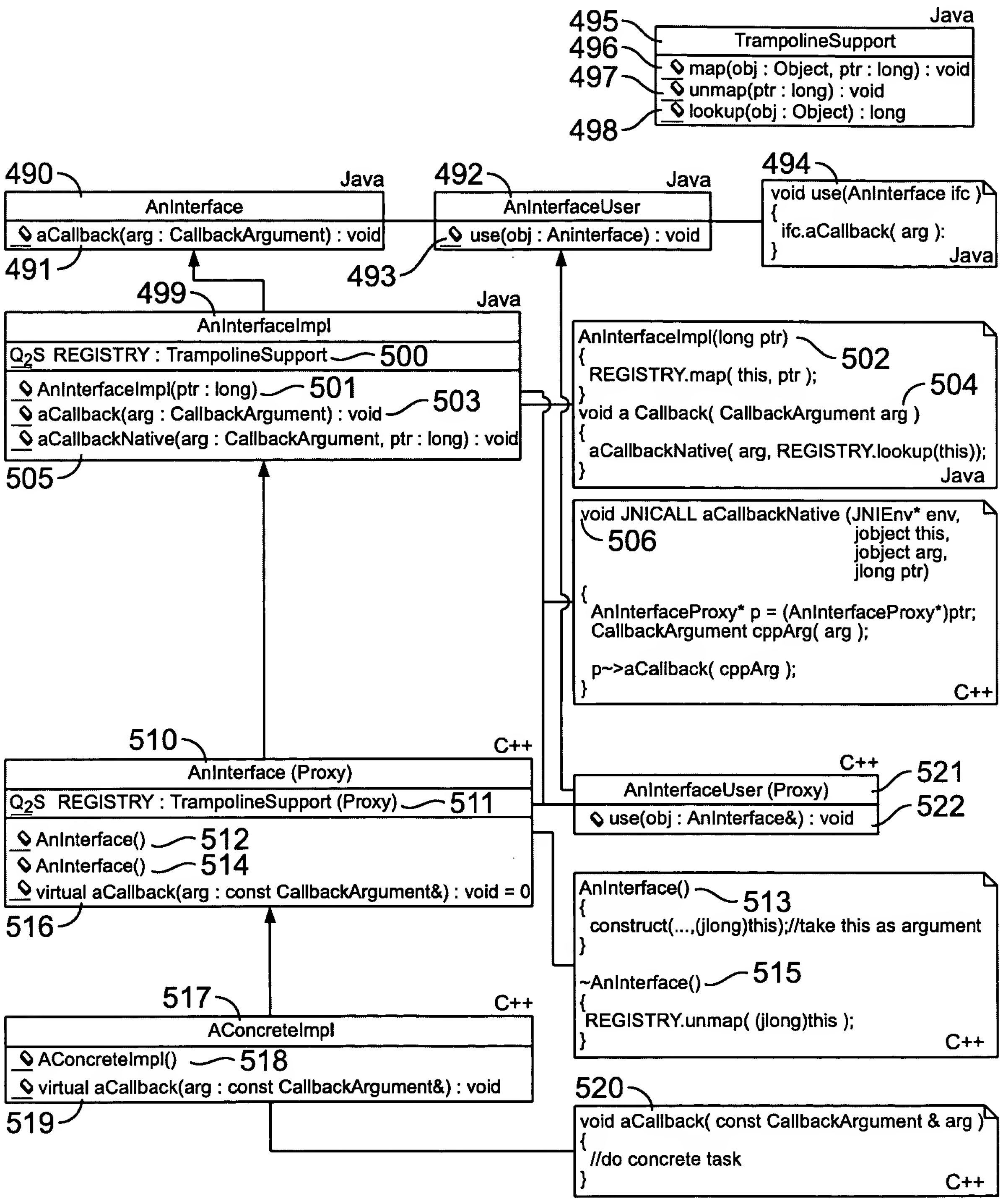

495
496
497
498
Java
TrampolineSupport
map(obj : Object, ptr : long) : void
unmap(ptr : long) : void
lookup(obj : Object) : long
490
Java
AnInterface
aCallback(arg : CallbackArgument) : void
491
492
Java
AnInterfaceUser
use(obj : Aninterface) : void
493
494
void use(AnInterface ifc)
{
ifc.aCallback(arg):
}
Java
499
Java
AnInterfaceImpl
Q2S REGISTRY : TrampolineSupport
500
AnInterfaceImpl(ptr : long)
501
aCallback(arg : CallbackArgument) : void
503
aCallbackNative(arg : CallbackArgument, ptr : long) : void
505
AnInterfaceImpl(long ptr)
502
{
REGISTRY.map(this, ptr);
}
504
void a Callback(CallbackArgument arg)
{
aCallbackNative(arg, REGISTRY.lookup(this));
}
Java
void JNICALL aCallbackNative (JNIEnv* env,
506
jobject this,
jobject arg,
jlong ptr)
{
AnInterfaceProxy* p = (AnInterfaceProxy*)ptr;
CallbackArgument cppArg(arg);
p~>aCallback(cppArg);
}
C++
510
C++
AnInterface (Proxy)
Q2S REGISTRY : TrampolineSupport (Proxy)
511
AnInterface()
512
AnInterface()
514
virtual aCallback(arg : const CallbackArgument&) : void = 0
516
C++
AnInterfaceUser (Proxy)
521
use(obj : AnInterface&) : void
522
AnInterface()
513
{
construct(...,(jlong)this);//take this as argument
}
~AnInterface()
515
{
REGISTRY.unmap((jlong)this);
}
C++
517
C++
AConcreteImpl
AConcreteImpl()
518
virtual aCallback(arg : const CallbackArgument&) : void
519
520
void aCallback(const CallbackArgument & arg)
{
//do concrete task
}
C++

FIG. 17

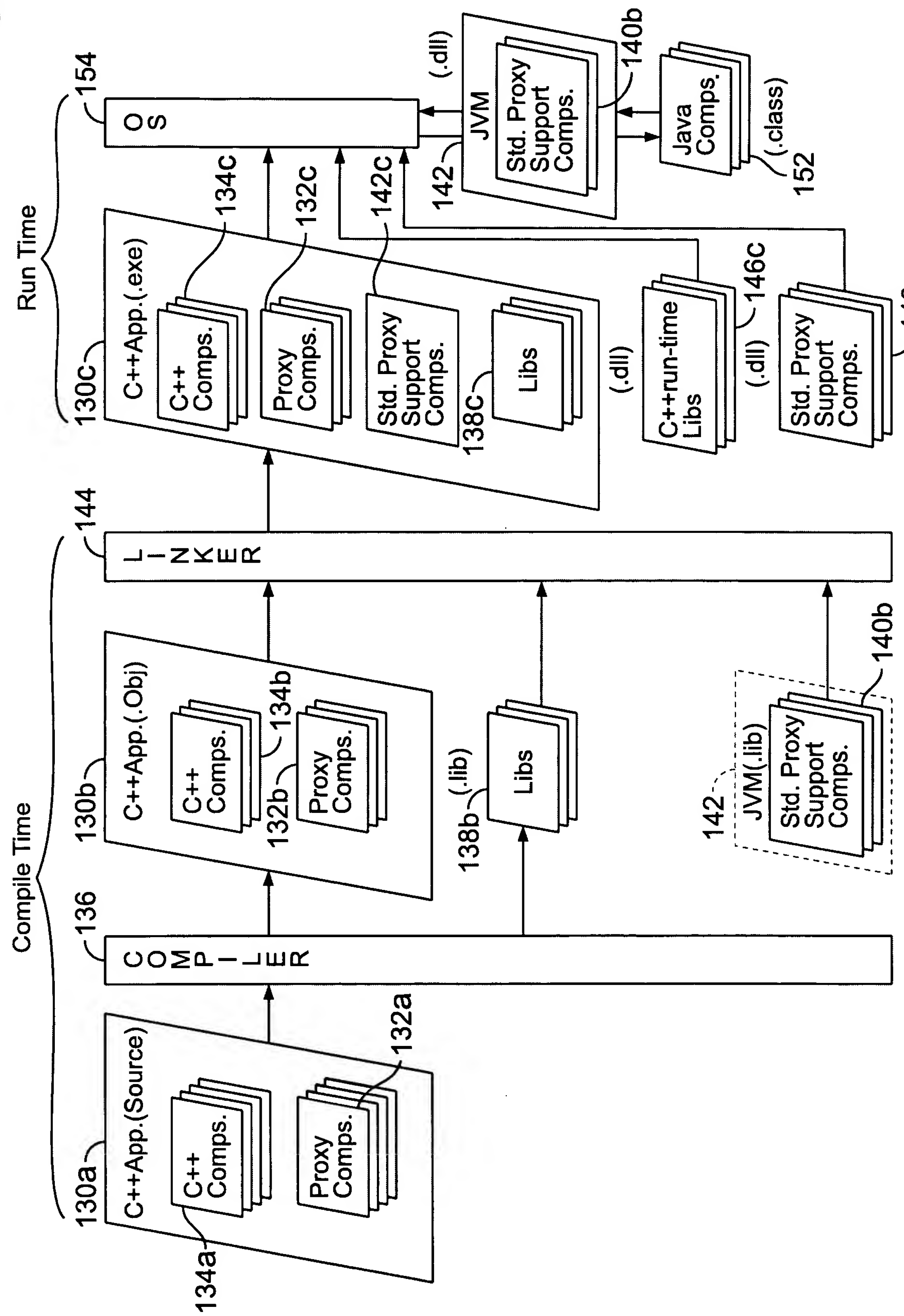

Compile Time
Run Time
130a
C++App.(Source)
C++ Comps.
Proxy Comps.
134a
132a
136
COMPILER
130b
C++App.(.Obj)
C++ Comps.
Proxy Comps.
134b
132b
138b
(.lib)
Libs
142
JVM(.lib)
Std. Proxy Support Comps.
140b
144
LINKER
130c
C++App.(.exe)
C++ Comps.
Proxy Comps.
Std. Proxy Support Comps.
Libs
138c
134c
132c
142c
(.dll)
C++run-time Libs
146c
(.dll)
Std. Proxy Support Comps.
148c
154
OS
142
JVM
(.dll)
Std. Proxy Support Comps.
140b
Java Comps.
(.class)
152

FIG. 18

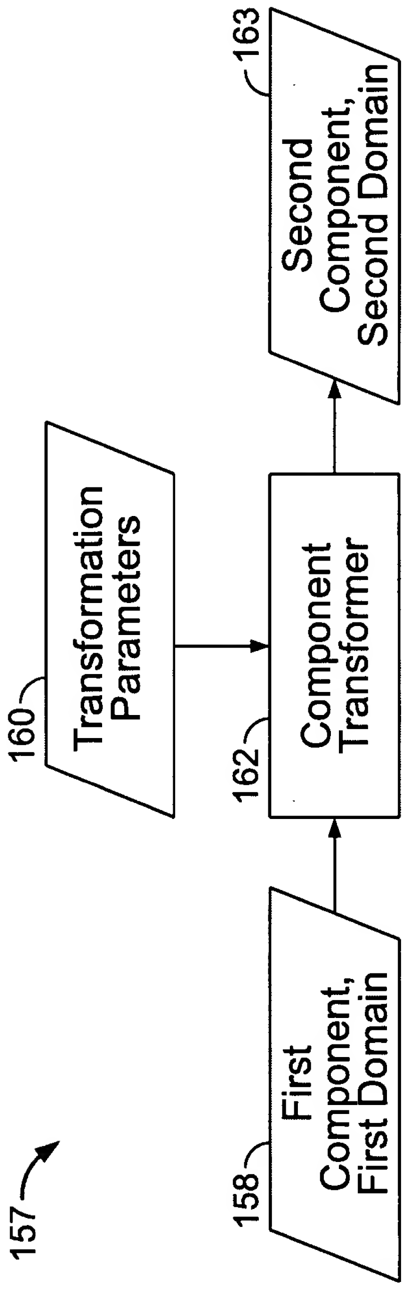

157
158
First Component, First Domain
160
Transformation Parameters
162
Component Transformer
163
Second Component, Second Domain

FIG. 19

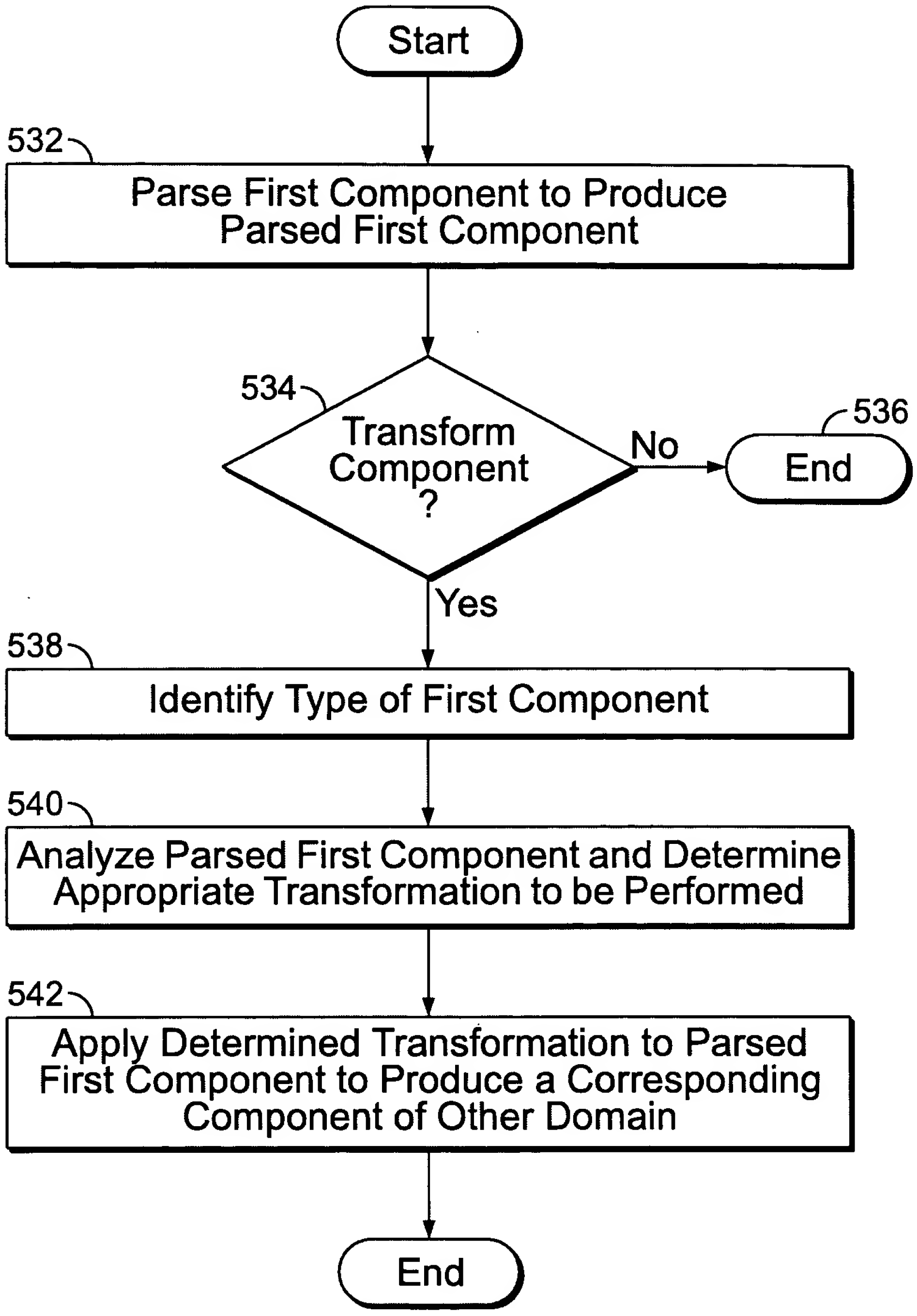

Start
532
Parse First Component to Produce Parsed First Component
534
Transform Component ?
No
536
End
Yes
538
Identify Type of First Component
540
Analyze Parsed First Component and Determine Appropriate Transformation to be Performed
542
Apply Determined Transformation to Parsed First Component to Produce a Corresponding Component of Other Domain
End

FIG. 20

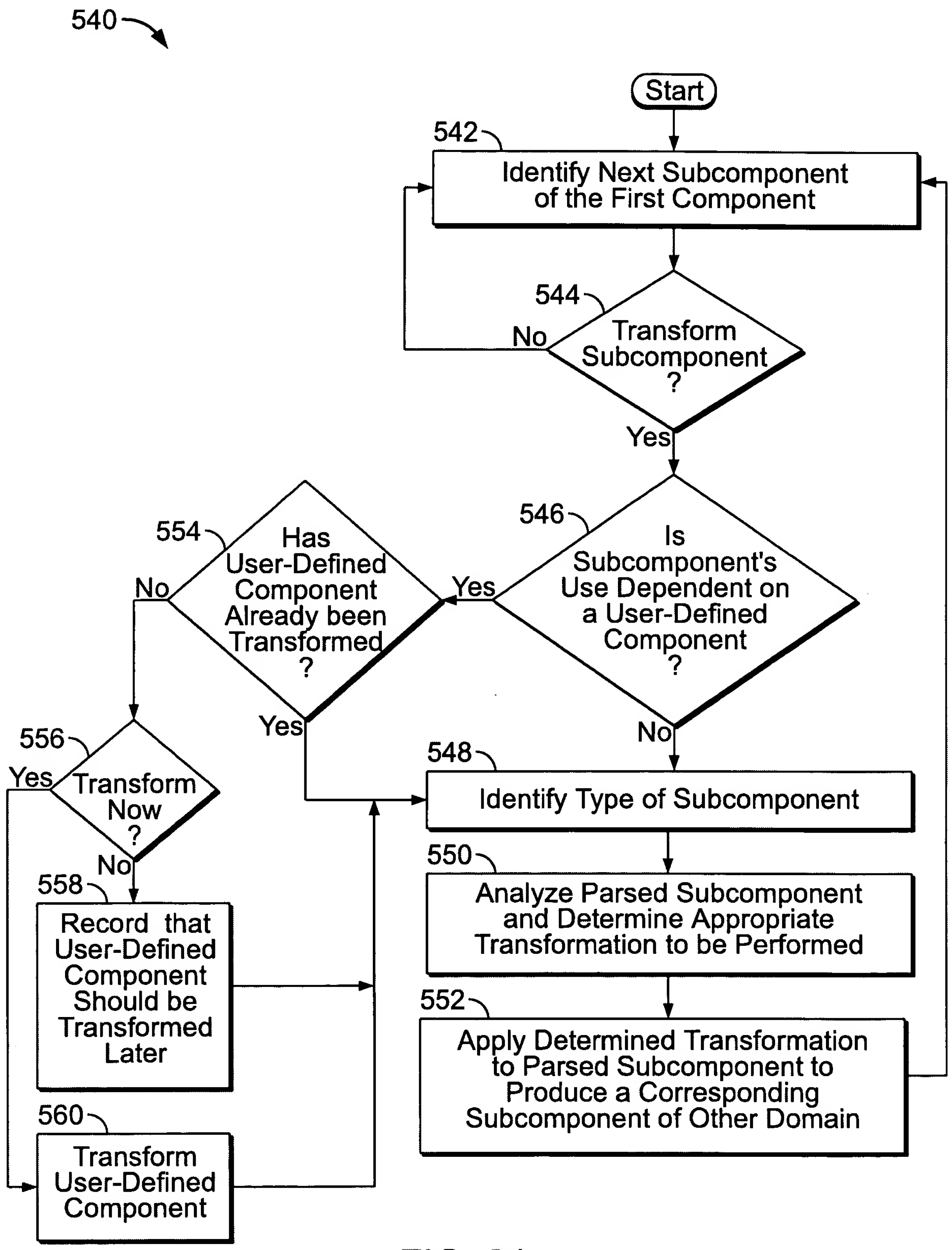

540
Start
542
Identify Next Subcomponent of the First Component
544
Transform Subcomponent ?
No
Yes
546
Is Subcomponent's Use Dependent on a User-Defined Component ?
Yes
No
554
Has User-Defined Component Already been Transformed ?
No
Yes
556
Transform Now ?
Yes
No
558
Record that User-Defined Component Should be Transformed Later
560
Transform User-Defined Component
548
Identify Type of Subcomponent
550
Analyze Parsed Subcomponent and Determine Appropriate Transformation to be Performed
552
Apply Determined Transformation to Parsed Subcomponent to Produce a Corresponding Subcomponent of Other Domain

FIG. 21

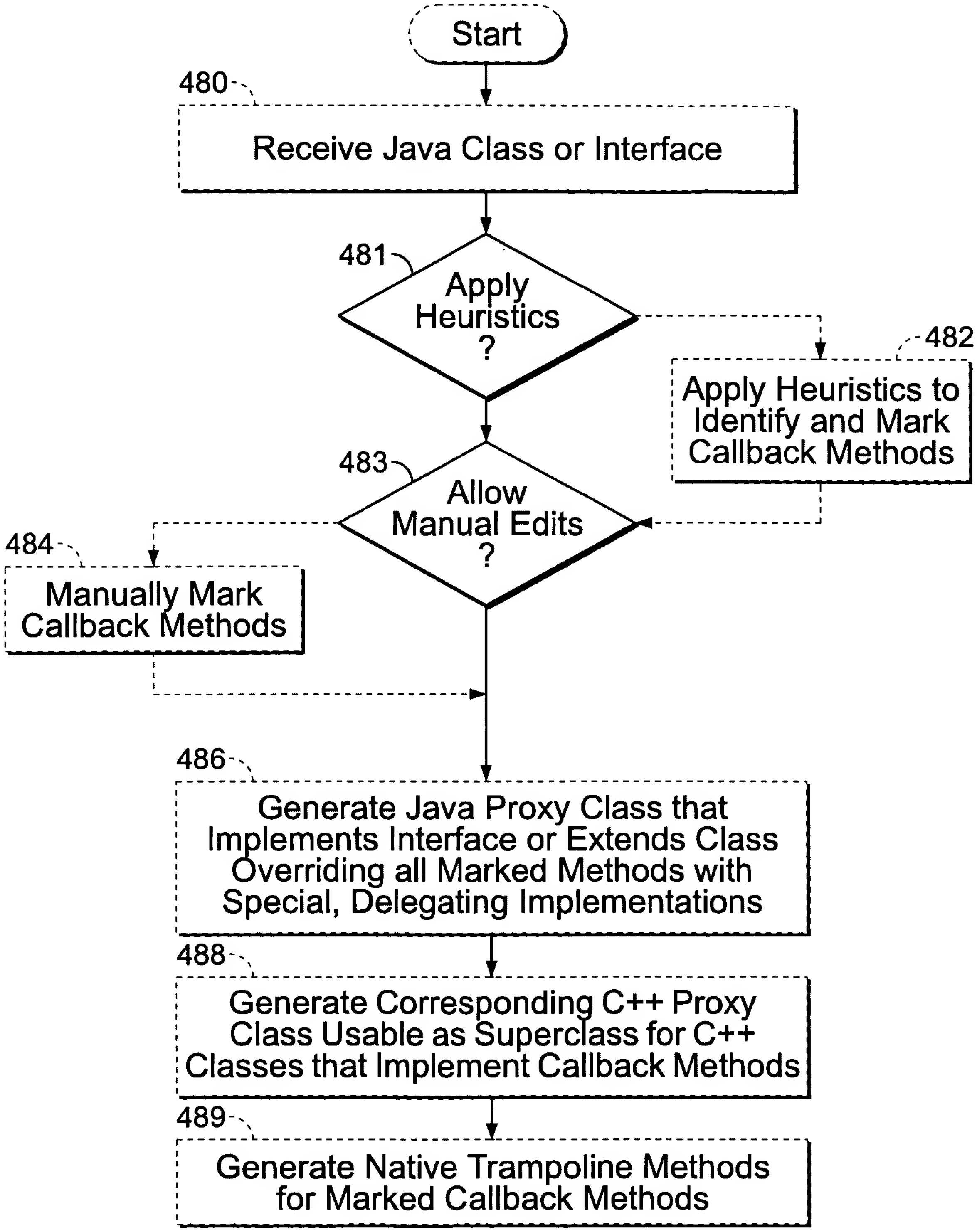

Start
480
Receive Java Class or Interface
481
Apply Heuristics ?
482
Apply Heuristics to Identify and Mark Callback Methods
483
Allow Manual Edits ?
484
Manually Mark Callback Methods
486
Generate Java Proxy Class that Implements Interface or Extends Class Overriding all Marked Methods with Special, Delegating Implementations
488
Generate Corresponding C++ Proxy Class Usable as Superclass for C++ Classes that Implement Callback Methods
489
Generate Native Trampoline Methods for Marked Callback Methods

FIG. 22

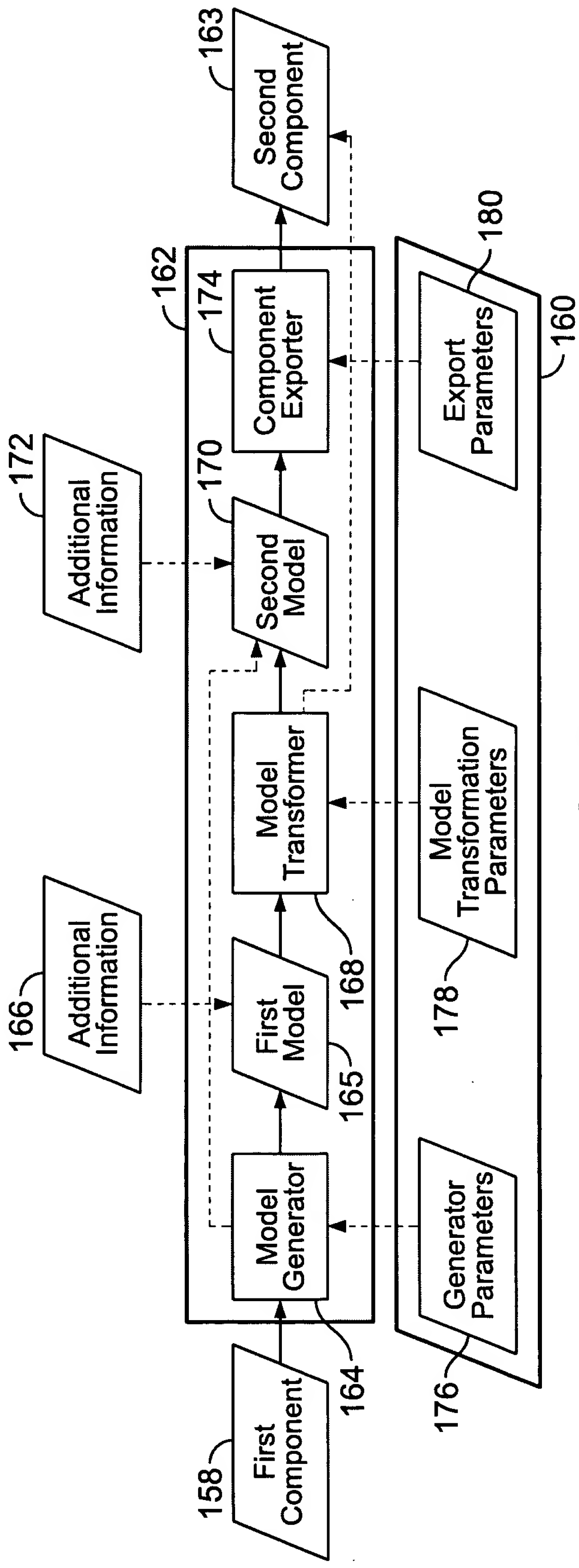
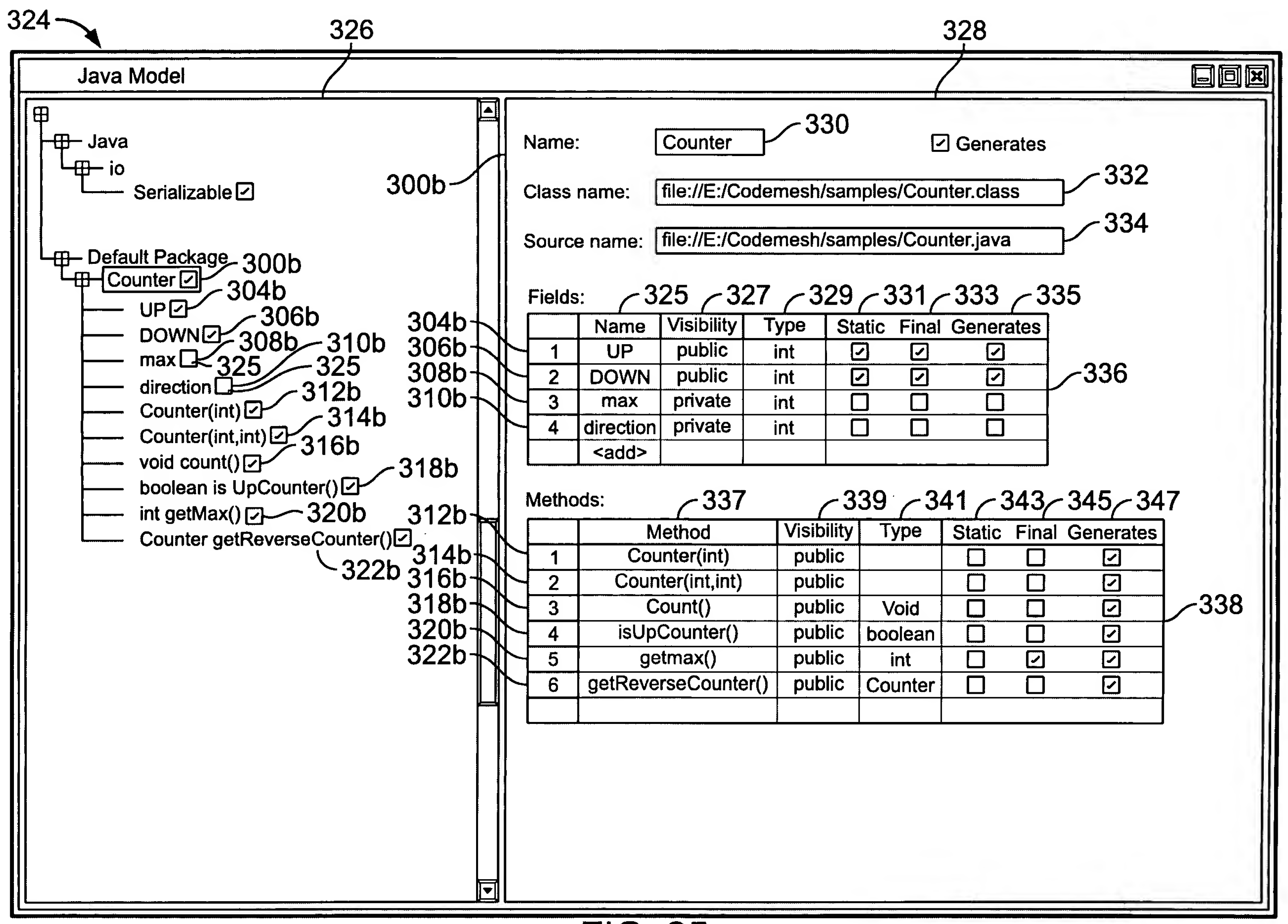

158
First Component
164
Model Generator
165
First Model
166
Additional Information
168
Model Transformer
170
Second Model
172
Additional Information
174
Component Exporter
162
163
Second Component
176
Generator Parameters
178
Model Transformation Parameters
180
Export Parameters
160

FIG. 23

```
300a
      public class Counter implements java.io.Serializable
      {                                    304a
         public static final int   UP = 1;
         public static final int   DOWN = 2;
                              308a        306a
         private int    max;
         private int    direction;
                              310a
         //creates a new UP-counter with the given maximum
         public Counter(  int _max  )
312a     {
                this(   _max, UP  );
         }

         //creates a new counter with given maximum and direction
         public Counter(  int _max, int _direction  )
314a     {
                 max = _max;
                 direction = _direction;
         }

         //counts in the direction specified and outputs the numbers
         public void    count()
         {
                 if(  direction == UP  )
                     for(  int I=0; I<max; I++  )
316a                     System.out.println (  " " + I  );
                 else if (  direction == DOWN  )
                       for (  int I=max-1;  I>=0;  I--  )
                            System.out.println(  " " + I  );
         }

         //returns true if this instance is an UP counter
         public boolean      isUpCounter()
318a     {
                 return  (  direction == UP  );
         }

         //returns the maximum of the counter
         public final int      getMax()
320a     {
                return max;
         }

         //creates a counter with the same maximum as this counter, but reverse direction
         public Counter      getReverseCounter()
322a     {
                return new Counter( max, direction  ==  UP ? DOWN : UP );
         }
      }
```

FIG. 24



Name: Counter ✓ Generates

Class name: file://E:/Codemesh/samples/Counter.class

Source name: file://E:/Codemesh/samples/Counter.java

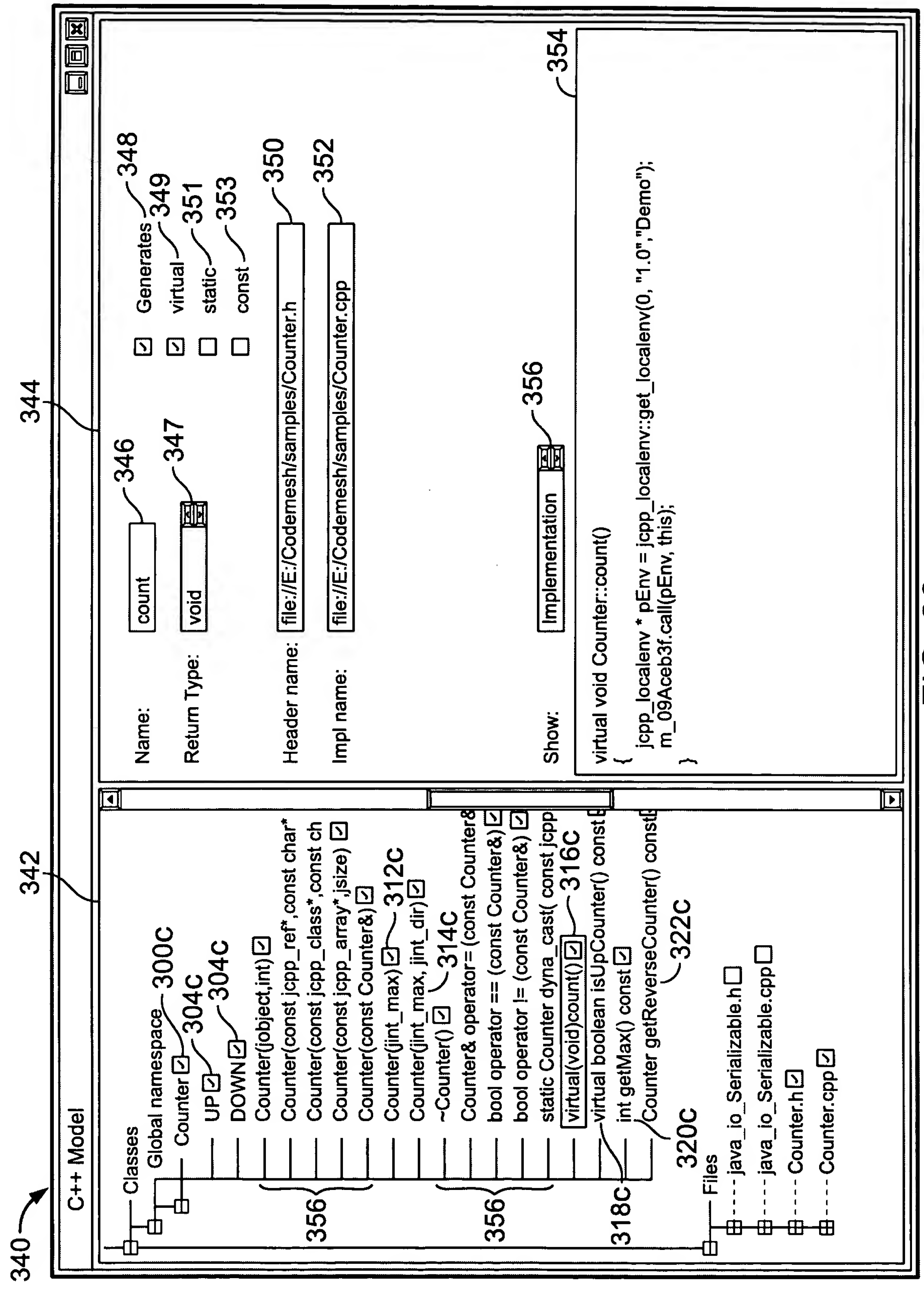
Fields:

	Name	Visibility	Type	Static	Final	Generates
1	UP	public	int	✓	✓	✓
2	DOWN	public	int	✓	✓	✓
3	max	private	int	☐	☐	☐
4	direction	private	int	☐	☐	☐
	<add>					

Methods:

	Method	Visibility	Type	Static	Final	Generates
1	Counter(int)	public		☐	☐	✓
2	Counter(int,int)	public		☐	☐	✓
3	Count()	public	Void	☐	☐	✓
4	isUpCounter()	public	boolean	☐	☐	✓
5	getmax()	public	int	☐	✓	✓
6	getReverseCounter()	public	Counter	☐	☐	✓

FIG. 25


340
342
344
C++ Model
Classes
Global namespace
Counter ✓
UP ✓
DOWN ✓
Counter(jobject,int) ✓
Counter(const jcpp_ref*,const char*
Counter(const jcpp_class*,const ch
Counter(const jcpp_array*,jsize) ✓
Counter(const Counter&) ✓
Counter(jint_max) ✓
Counter(jint_max, jint_dir) ✓
~Counter() ✓
Counter& operator=(const Counter&
bool operator == (const Counter&) ✓
bool operator != (const Counter&) ✓
static Counter dyna_cast(const jcpp
virtual(void)count() ✓
virtual boolean isUpCounter() const
int getMax() const ✓
Counter getReverseCounter() const
Files
java_io_Serializable.h
java_io_Serializable.cpp
Counter.h ✓
Counter.cpp ✓
300C
304C
304C
312C
314C
316C
318C
320C
322C
356
356
Name:
Return Type:
Header name:
Impl name:
Show:
346
347
348
349
350
351
352
353
354
356

Name: count

Return Type: void

✓ Generates
✓ virtual
☐ static
☐ const

Header name: file://E:/Codemesh/samples/Counter.h

Impl name: file://E:/Codemesh/samples/Counter.cpp

Show: Implementation

```
virtual void Counter::count()
{
  jcpp_localenv * pEnv = jcpp_localenv::get_localenv(0, "1.0","Demo");
  m_09Aceb3f.call(pEnv, this);
}
```

FIG. 26

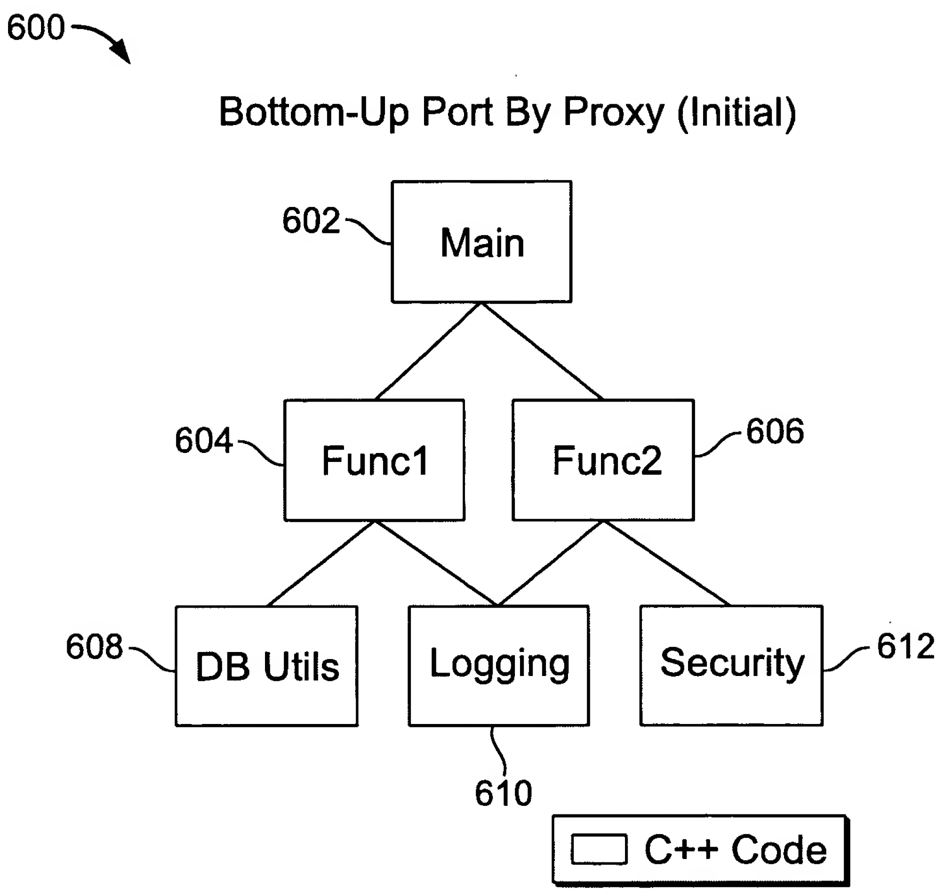

600
Bottom-Up Port By Proxy (Initial)
602
Main
604
Func1
Func2
606
608
DB Utils
Logging
Security
612
610
C++ Code

FIG. 27A

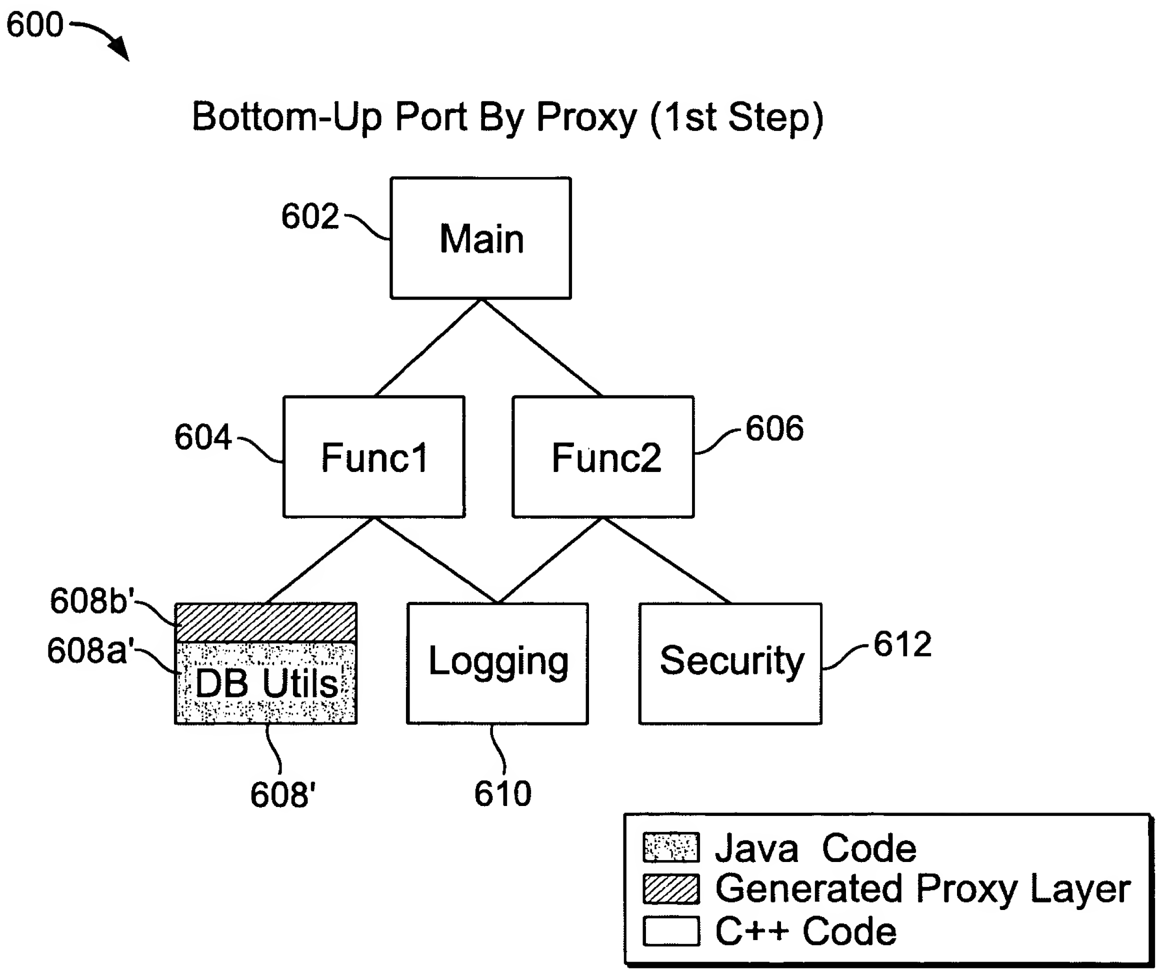

600
Bottom-Up Port By Proxy (1st Step)
602
Main
604
Func1
Func2
606
608b'
608a'
DB Utils
Logging
Security
612
608'
610
Java Code
Generated Proxy Layer
C++ Code

FIG. 27B

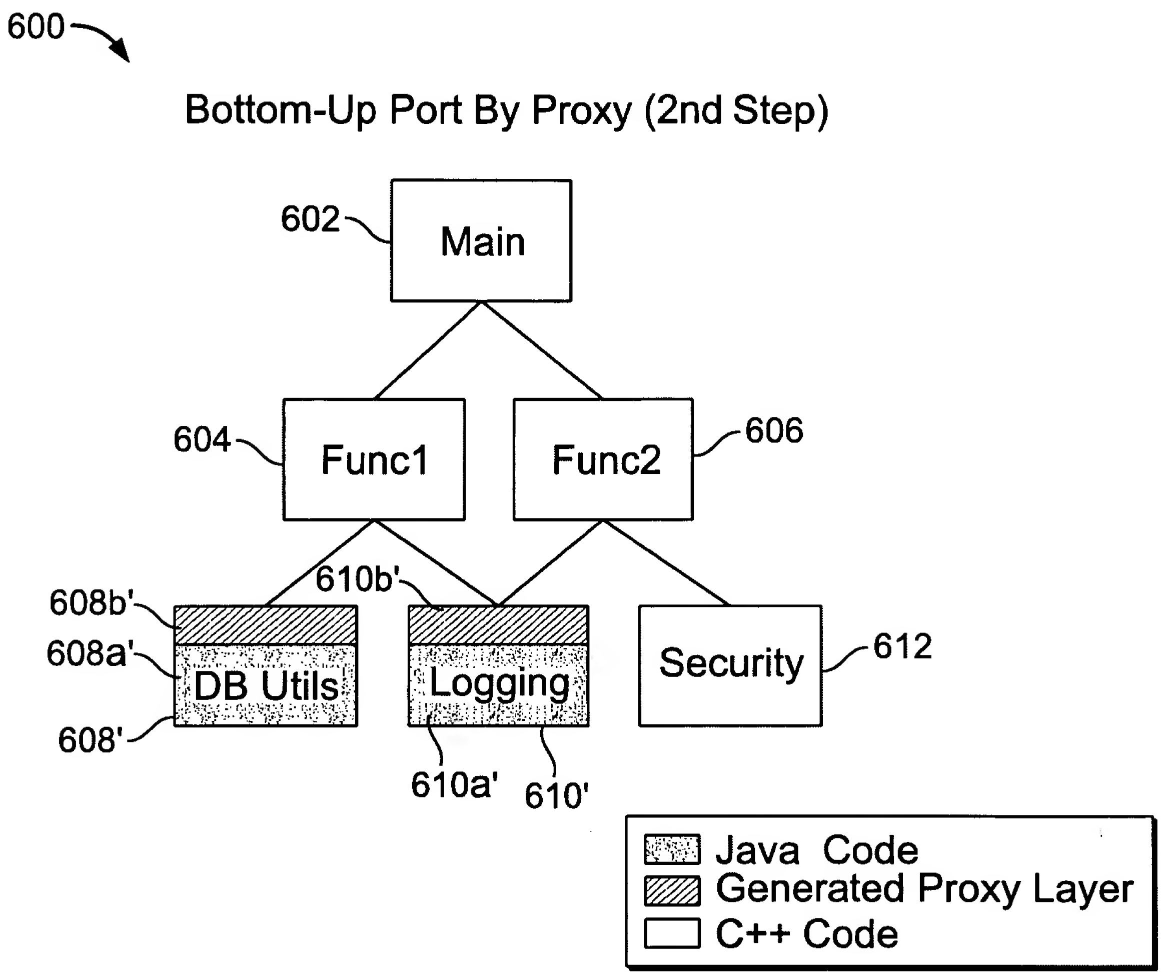

600
Bottom-Up Port By Proxy (2nd Step)
602
Main
604
Func1
Func2
606
608b'
610b'
608a'
DB Utils
Logging
Security
612
608'
610a'
610'
Java Code
Generated Proxy Layer
C++ Code

FIG. 27C

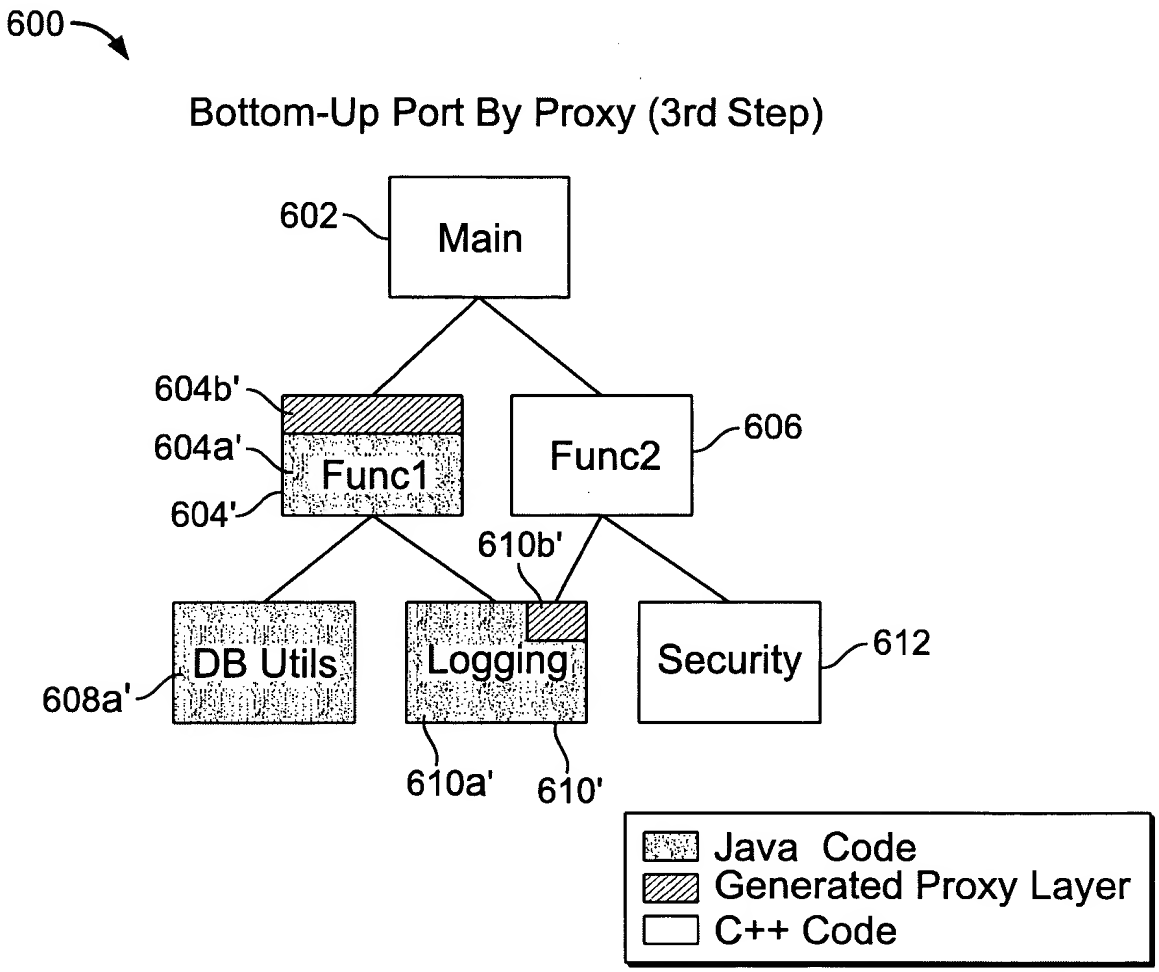

600
Bottom-Up Port By Proxy (3rd Step)
602
Main
604b'
604a'
604'
Func1
Func2
606
610b'
DB Utils
608a'
Logging
610a'
610'
Security
612
Java Code
Generated Proxy Layer
C++ Code

FIG. 27D

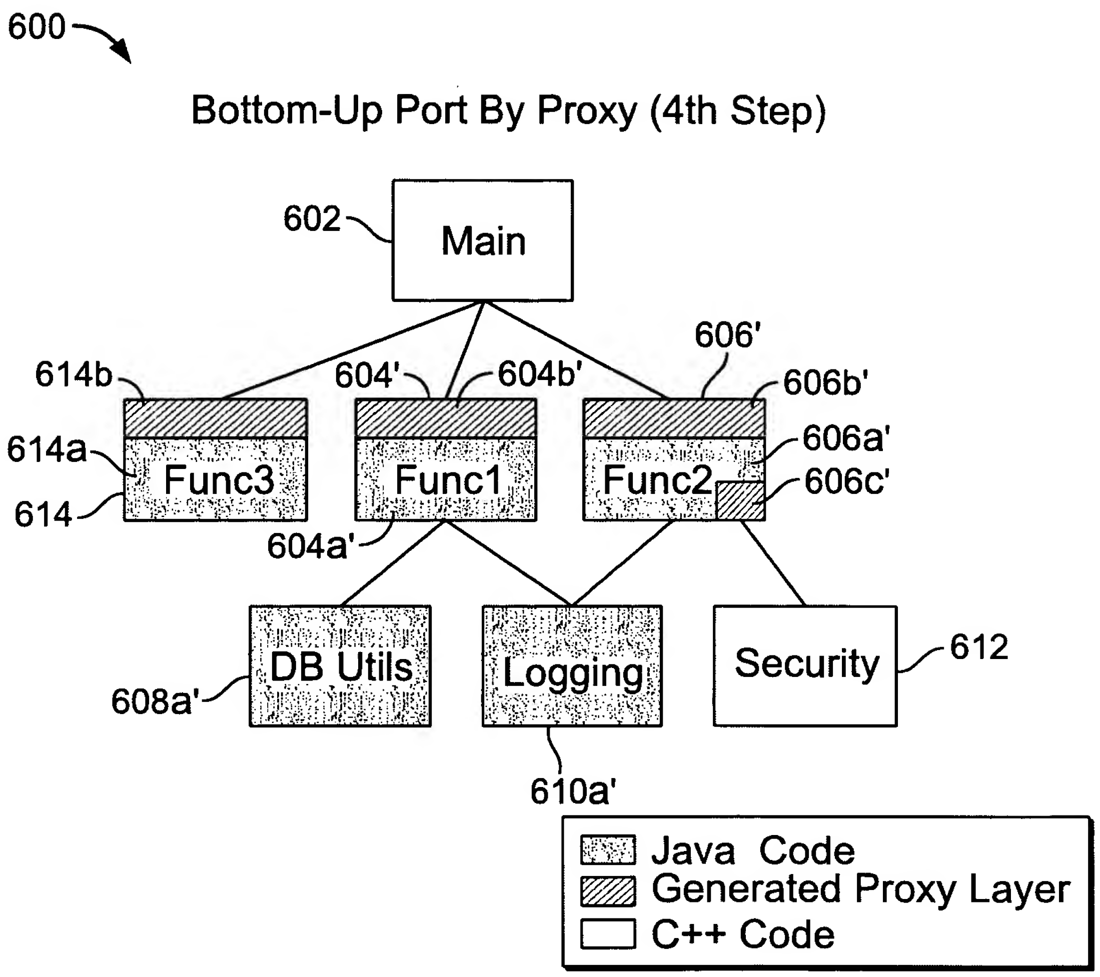

600
Bottom-Up Port By Proxy (4th Step)
602
Main
614b
604'
604b'
606'
606b'
614a
Func3
Func1
Func2
606a'
606c'
614
604a'
608a'
DB Utils
Logging
Security
612
610a'
Java Code
Generated Proxy Layer
C++ Code

FIG. 27E

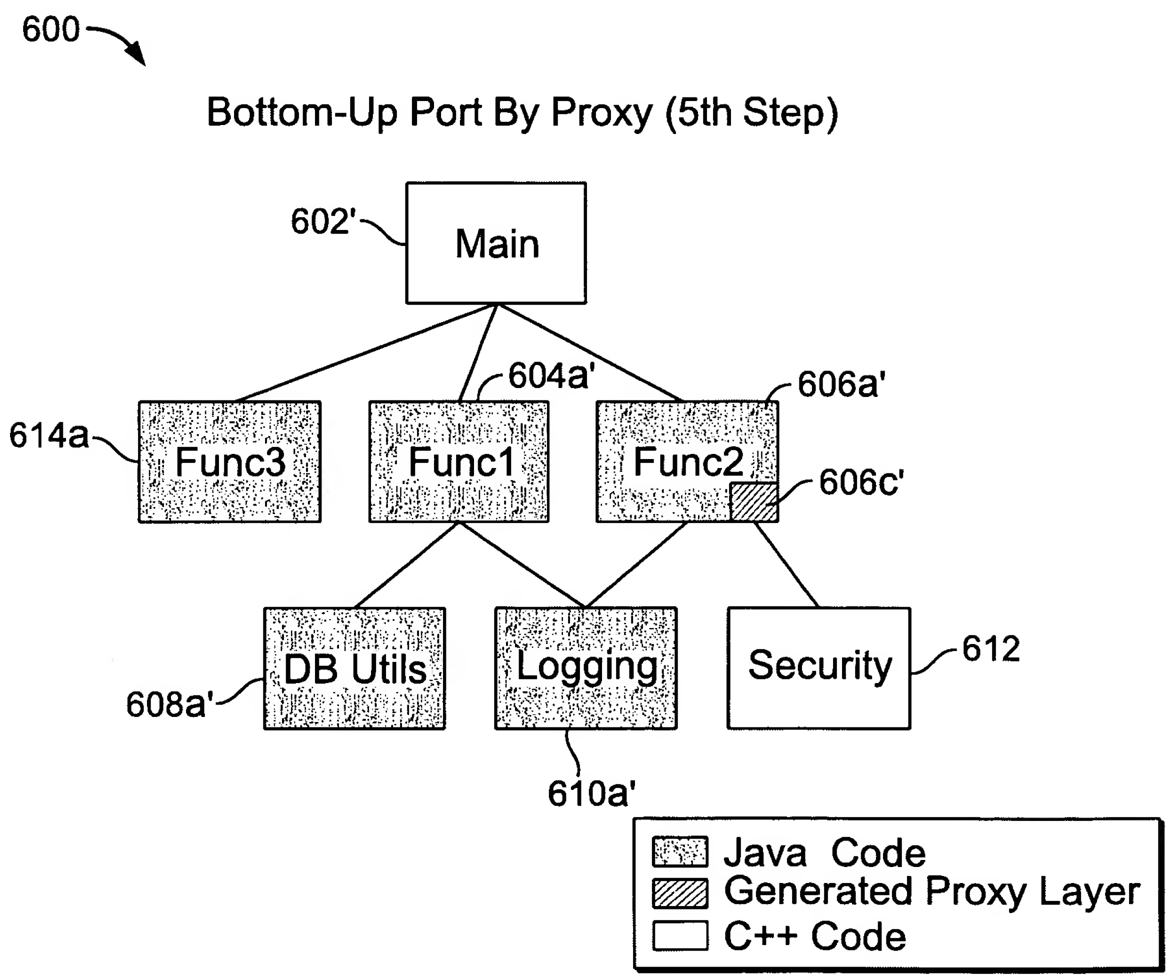

600
Bottom-Up Port By Proxy (5th Step)
602'
Main
614a
Func3
Func1
604a'
Func2
606a'
606c'
DB Utils
608a'
Logging
610a'
Security
612
Java Code
Generated Proxy Layer
C++ Code

FIG. 27F